TAMPEREEN TEKNILLINEN YLIOPISTO

TAMPERE UNIVERSITY OF TECHNOLOGY
Faculty of Computing and Electrical Engineering

Manuele Cucchi

DESIGN AND IMPLEMENTATION OF A FFT PRUNING ENGINE FOR DSA-ENABLED COGNITIVE RADIOS

Master of Science Thesis

Subject approved by Faculty Council
March 6th, 2013
Examiners: Prof. Jari Nurmi (TTY)
MSc Roberto Airoldi (TTY)

To my family …

i. Contents

ii. List of Figures

iii. Acknowledgement

With these few words, I would like to thank all the people who have been close in recent years. Thanks to their support I managed to complete this first important step of my life.

The first thought is for my family, to my mother Letizia, my father Marco, my sisters Laura and Beatrice, my brother Michele and, not least, to my girlfriend Giulia. They gave me the strength to go forward and the support to believe that I would be able to do it, which has always made me felt valued. I believe that self-esteem is important as well as the preparation that occurs in a certain field, because it helps you to make the most difficult thing of all, namely be resourceful and jump into the new reality. This is why I thank you sincerely.

A special thought to all those who represent for me the escape from certain contexts. I thank my friends Gianluca, Alessandro, Alberto, Mattia, Fabio, Nicolas and all those who were close to me during my path.

I also would like to thank all my classmates whereby I have established incredible relationships facts of hours on Skype to prepare exams and nice "pause moments" between lessons. Thank you guys!

My last thought, but not least important, goes to those who have been for me an important reference in this recent experience abroad. I thank my supervisor Roberto Airoldi for his experience and his useful support full of advices, and the professors Aldo Romani and Jari Nurmi for giving me the opportunity to work with them.

Manuele Cucchi

iv. Abstract

Tampere University of Technology
Manuele Cucchi: DESIGN AND IMPLEMENTATION OF A FFT PRUNING ENGINE FOR DSA-ENABLED COGNITIVE RADIOS
MSc Thesis, 78 pages
March 2013
Major: Digital and Computer Systems
Examiners: Prof. Jari Nurmi, MSc Roberto Airoldi
Keywords: DSA, Cognitive Radio, NC-OFDM, PRUNING, FPGA

. . .

Dynamic spectrum access (DSA) communications are an efficient way to improve the efficiency of spectrum utilization in the context of cognitive radio networks. In fact, through the utilization of DSA-enabled communications, secondary users are able to utilize portions of spectrum that are currently left unused by primary users both in time and space domains. However, the available spectrum is highly scattered throughout the frequencies and therefore agile transmission techniques have to be used to efficiently allocate the secondary user's communications in the available spectrum gaps. In this context, Non-Contiguous OFDM (NC-OFDM) and Discontinuous-OFDM (D-OFDM) systems are feasible solutions for fully exploit the available spectrum, even if highly scattered. NC-OFDM, as well as D-OFDM, is able to turn on and off its sub-carriers allocations, in order to place the transmitted data in the unused spectrum gaps, without interfering with the communications of primary users. From an architectural point of

view, the utilization of NC-OFDM enables to improve the energy efficiency of the (de)modulation block. In fact, the computation of FFT/IFFT can be simplified via the utilization of pruning algorithms. Pruning algorithms are able to eliminate at run-time the computation of dummy operations, such as the addition or the multiplication of zero terms, reducing the complexity of the original algorithm. At receiver side the unused sub-carriers are considered as zero-value inputs in the FFT block. Thus, in a scenario where the number of unused sub-carrier is significant, FFT pruning is able to lighten the computation of the FFT, enabling a more efficient implementation of the algorithm.

This research work presents the design and implementation of a FFT pruning block, which is an extension to the FFT core for OFDM demodulation, enabling run-time pruning of the FFT algorithm, without any restrictions on the distribution pattern of the active/inactive sub-carriers. The design and implementation of FFT processor core is not the part of this work. The whole design was prototyped on an ALTERA STRATIX V FPGA to evaluate the performance of the pruning engine. Synthesis and simulation results showed that the logic overhead introduced by the pruning block is limited to a 10% of the total resources utilization. Moreover, in presence of a medium-high scattering of the sub-carriers, power and energy consumption of the FFT core were reduced by a 30% factor.

CHAPTER 1

Cognitive Radio environment

This first chapter gives a brief presentation about the context of our work, and the motivations behind it. The concept of *Dynamic Spectrum Access [DSA]*, and the typical technique of receiving a transmission in a *NC-OFDM* and *OFDM* systems will be introduced; in particular, the block Fast Fourier Transform (FFT) and theory regarding the calculation of Radix-2 FFT algorithms will be presented in a comprehensive manner. Finally, the state of art related to the work done and the highlight the differences with this work will be present.

1.1 Dynamic Spectrum Access and concept of Cognitive Radio

The spectrum of the electromagnetic waves is defined as the range of all possible electromagnetic radiation and it is used by all transmitters and receivers that are coordinated by national law. The partition of spectrum most used today, is between 30 MHz and 10 GHz where are allocated bands for AM and FM radio, analog and digital televisions, mobile phones (e.g. *GSM* and *UMTS*), wireless networks such as Bluetooth and Wi-Fi, and satellite broadcasts.

The field of telecommunications is constantly evolving in terms of new technologies and emerging systems, especially wireless, bordering on the market since the increasing of the demand from the users of mobile services. Meeting the needs of these technologies and

their services we need to face the problem of spectral resource. The regulation of use of the spectrum provides an fixed assignment of frequency bands to all wireless systems, and imposes a limit of maximum transmission power, in order to avoid interference with other users present at a great distance.

In recent years, the exponential growth of wireless systems and mobile services, has led to an increasing demand for access to the spectrum in certain bands paving the way to the problem known as spectrum scarcity[1].

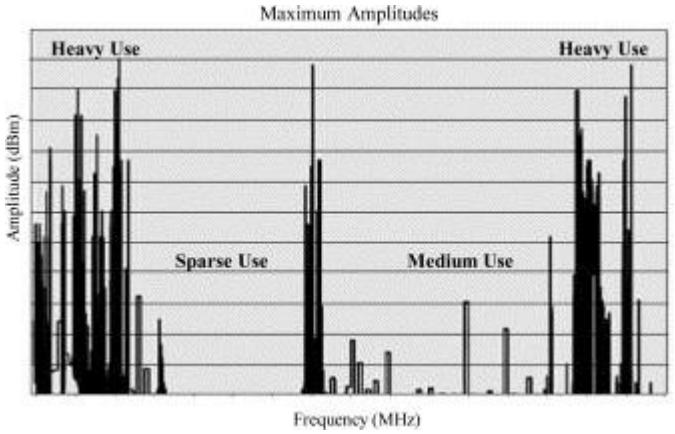


Figure 1.1: Example of typical spectrum occupancy [2].

Nevertheless, as shown in Figure 1.1 in many geographical areas, the average factor of usage of some bands turns out to be low, due to these services which are underused in many cases. In 1994 *P.Baran* suggested the need to redraw the political spectrum allocation, opening the door for a more innovative wireless technology based on the concept of open spectrum, known also as ***Dynamic Spectrum Access (DSA)*** *[3]*. An important prerequisite of such technologies is the ability to observe the environment and take decisions in order to adapt quickly to the conditions of the transmission.

An American national organization called *Federal Communications Commission* (FCC) investigated the effective exploitation of the spectrum, taking the example of some U.S. cities such as Atlanta and New Orleans [4]. Among the main results of the study of spectral occupancy, it emerged that many parts of the spectrum are not used continuously over time, caused by a complicated frequency management and regulation of spectrum

licenses. In other words, in most cases an inefficient use of the spectrum in frequency, space and time, with the emergence of an increasing number of mobile communications equipment presents itself as an emerging problem of scarcity spectral. According to the FCC, the solution to this problem takes comfort behind the definition of Cognitive Radio (CR) introduced by Mitola [5].

CR indicates a wireless system with awareness of the surrounding environment, able to intelligently manage the radio resources, and provide resources and the most appropriate wireless services for the communication needs of the user. Many people have given a different definition of CR: the FCC established as Cognitive Radio as any radio device that can adapt to the available spectrum, changing their transmission parameters on the basis of the surrounding environment [6].

Given the definition of Cognitive Radio, the idea to solve the problem of spectrum shortage is based on a hierarchical structure of access to spectrum, with the presence of primary users, the license holder, and the secondary users without license that can be considered opportunistic users.

Because of the intelligence of these devices, it is possible to deactivate the spectrum (or parts of it) and detect the presence of primary communications or identify any white space, also called “holes” in the spectrum; with white spaces are indicated all those parts of the spectrum free from any communication. Thus, unlicensed users (secondary) have the possibility to use the portions of the spectrum available with a limit on the interference perceived by primary users, creating a secondary market of users who have less priority and then improve spectrum sharing. Hence, cognitive radios could be used in many contexts to improve the exploitation of the spectrum from the users by reducing the part of unused bands.

In order to achieve the proposed results, a CR technology will have to rely on real time highly reconfigurable wireless platforms that can adapt in real time to changes of the environment. Therefore, the device used in this technology requires the presence of a software programmable interface of a receiver of Cognitive Radio (call also Software Define Radio), and the use of processors like DSP and FPGA, replacing dedicated hardware, able to perform real-time sensing operations of the channel are indispensable.

The use of DSP and FPGA is already an established reality for the processing of baseband signal, but it can't be yet considerate as "SDR systems", because not all baseband functions are implemented on this architecture and, furthermore, the software is limited and loaded before. Therefore, the entire system remains bound to a specific type of radio interface, without any possibility of reconfiguration. These systems do not exhibit a sufficient spectral agility level to implement a receiver of Cognitive Radio.

1.2 OFDM System

The Orthogonal Frequency Division Multiplexing (OFDM) modulation is the basic technique used in the physical of many wireless communication systems. OFDM is a form of numerical multicarrier modulation where the transmitted information is spread across N subcarriers, orthogonal to each other and spaced in frequency by a multiple of $\Delta f = 1/\Delta t$ (called modulation frequency) [7]. The relation of orthogonally between the subcarriers is used to cancel the interference between the different flows received during the phase demodulation, said ISI (*InterSymbol Interference).* The binary input stream of with bit rate R_b (and bit duration $T_b = 1/R_b$) is transmitted in blocks on the channel where each block is associated with an "*OFDM symbol* ". With *N = number of the carriers* and $m = number\ of\ bits$ then each block is composed by $N \cdot m$ *[bits],* and there are $M = 2^m$ elementary symbols of the numerical modulation used by each carrier. In the other words, for each carrier there is a type of digital modulation of M signals. For example, the most commonly used modulations are *BPSK, QPSK, 16-QAM, 64-QAM, 128-QAM, 256-QAM* ($m = 1,2,4,6,7,8$). Thus, the signal transmitted on the channel is the sum of a large number of sinusoidal carriers, arbitrary modulated with the phase and amplitude.

The OFDM technique derives directly from the FDM (Frequency Division Multiplexing). However, the distinction of the various received signal, is achieved by placing the channels at a distance such that the spectrum of the signals does not overlap. Instead, in an OFDM system, because of the fact that the subcarriers are orthogonal to each other, the subcarriers can be placed also with overlap portions of the sidebands of the individual

sub-channels still obtain a correct reception of signals. Therefore, there is a considerable a bandwidth saving compared to the case FDM, and as consequence more channels and then users can be inserted into same portion of the spectrum. In order to understand better this concept the Figure 1.2 is shown:

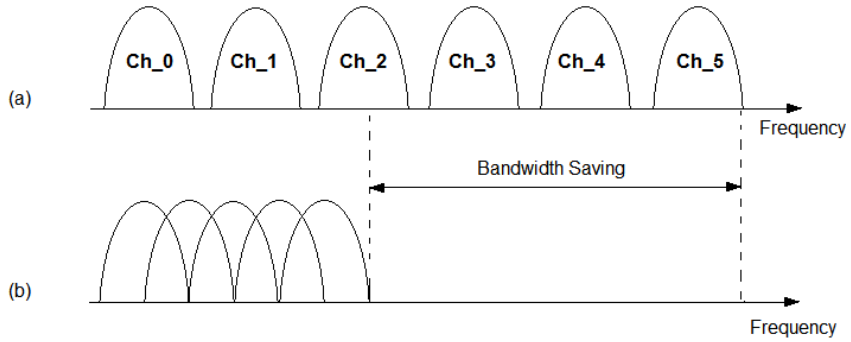


Figure 1.2: (a) conventional FDM modulation,, (b) OFDM modulation.

Since there is a parallel transmission of N carriers, the OFDM symbol time T_s, (constituted by N elementary numeric symbols), is related to the bit time as follow*:* $T_s = N \cdot (T_b \cdot m)$ where $T_s \gg T_b$.

Similarly, for the symbol rate: $R_s = \frac{R_b}{N \cdot m}$ [symbols/sec] where $R_s \ll R_b$.

The famous formula of the transmitted signal of OFDM modulation technique is :

$$s(t) = \sum_{i=0}^{N-1} s_i \, g(t) \, e^{j(2\pi f_i t)} \qquad (1)$$

From equation (1), a great analogy with the definition of Inverse Discrete Fourier Transform (IDFT) can be observed:

$$x[n] = \sum_{i=0}^{N-1} X[i] \; e^{j\left(\frac{2\pi i}{T}\right)} \qquad (2)$$

Typical OFDM modulator and demodulator schemes can be implemented via software and hardware through FFT systems [8]. In Figure 1.3, a typical transmitter side is shown: the flow of data produced by the source is divided into blocks and serial/parallel selector divides the flow of information in parallel flows called subcarriers.

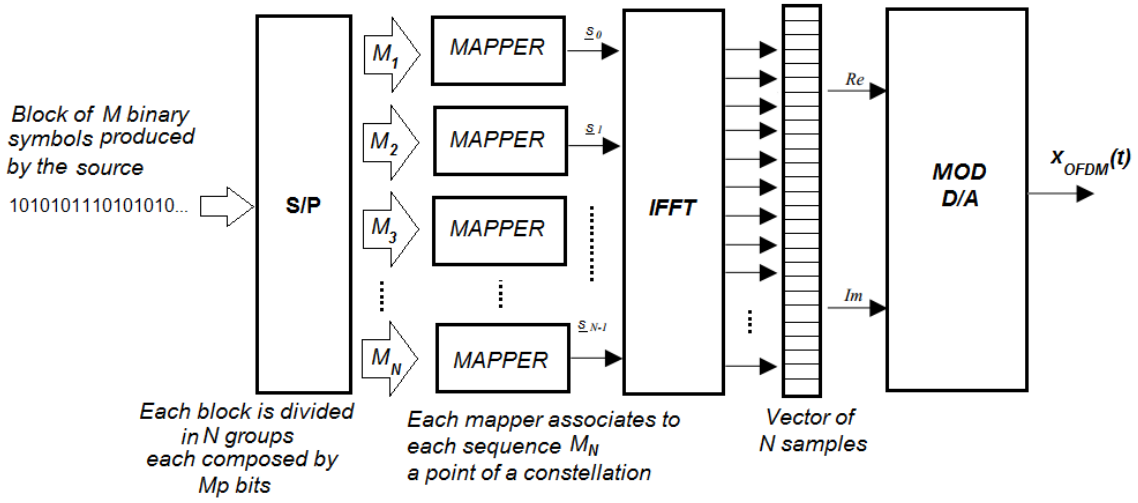


Figure 1.3: OFDM transmitter scheme.

Per each subcarrier, a mapper block associates a point of a constellation. Each constellation is composed by $L = 2^M$ where M is the length of each block produced by the source.

An IDFT block performs the operation of Inverse Fast Fourier Transform (IFFT). The result of the IDFT is a sequence of complex coefficients. D/A block and a modulator block will generate different subcarriers. To better understand the task of the block MAPPER, Figure 1.4 shows the 16-QAM modulation. The QAM modulation is a digital modulation in which the input signal is split in different subcarriers with different amplitude. In addition, the modulated signals are added together and a waveform that is a combination between the phase and of the amplitude modulation can be obtained. A sequence of four bits corresponds to each point of the constellation.

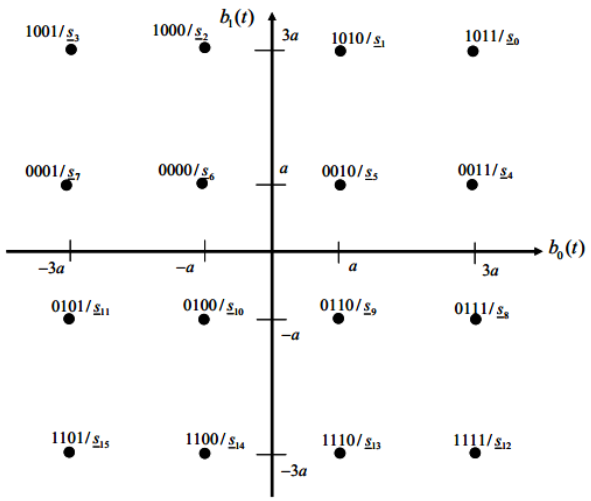


Figure 1.4: 16-QAM Modulator[9].

Therefore, every 4-bit inputs of the modulator are associated with a symbol in the constellation. Similarly, in reception a DFT algorithm is used and the correlation values with the center frequency of each subcarrier is calculated. Thus, the transmitted data can be reconstructed without crosstalk and without causing the disadvantageous inter-carrier.

After the FFT block, a de-mapper will associate the received symbol with the nearest constellation point and it will describe each symbol with M_N bits; the final result is again serialized to produce the M-bit (originally symbol).

This architecture scheme can be realized through the use of DSP or FPGA. The implementations are based on the structure of the FFT calculation called “butterfly” and will be presented in more detail in the next chapter.

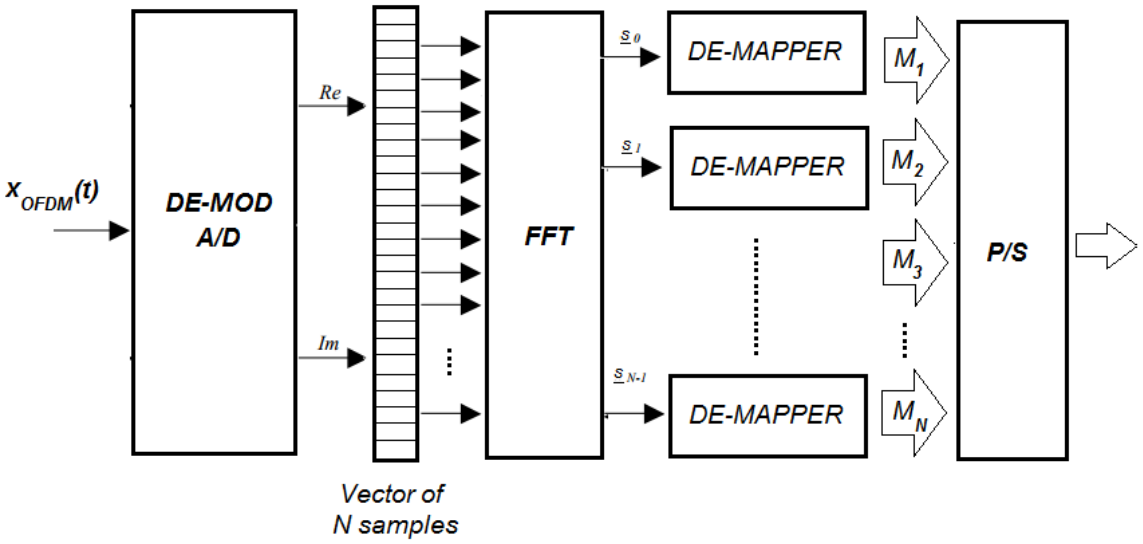


Figure 1.5: OFDM receiver scheme.

1.3 NC-OFDM System

CR technology requires, as main characteristic, the possibility of developing transmission systems with a sufficient degree of spectral agility. Hence, secondary user are able to use spectral gaps left available by primary users. With the aim of achieving such a degree of bandwidth usage, a new modulation technique is introduced: the *Non-Contiguous-OFDM [10]*.

NC-OFDM is a technique of modulation and demodulation based on the theoretical concepts of the OFDM technique, which presents a DFT core for the calculation and introduce a considerable simplification in the process of demodulation of the received signals. In a NC-OFDM systems the information flow can be divided into more information flows associated at different subcarriers but, respect classic OFDM, not necessarily contiguous. It follows that the subcarriers can be deactivated for different portion of spectrum, called "holes spectrum" based on the decision of the user that wants

to transmit. A user who wants to transmit can decide, for example, to distribute his information using the short and scattered parts of the spectrum free.

Figure 1.6 shows an example of NC-OFDM transmission with a part of subcarriers disabled:

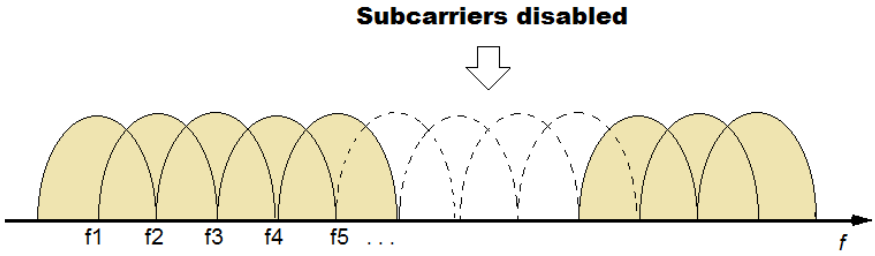


Figure 1.6: Example of NC-OFDM transmission.

In order to not miss the information during the transmission phase, the information flow is consequently adapted to be transmitted in the remaining active sub-carriers. The system can dynamically decide to "prune" an arbitrary number of subcarriers to give the possibility to an user to occupy a certain portion of the spectrum and decide to deactivate some carriers on the basis of communication parameters (for example interfere with other communications).

Thus, to take full advantage of the DFT hardware resources, there is a different way to define the demodulator. As described in [23], a typical modulator/demodulator NC-OFDM is shown in Figure 1.7: the input data *x (n)* is modulated and divided in a parallel flux through a serial/parallel (S/P) block. Each converted data is assigned to a subcarrier that is processed by a IFFT. Through a spectrum sensing phase, the transmitter can decide which subcarrier to turn off. Hence, the block cyclic prefix appends a "bit range" in order to avoid Inter-Symbol Interference.

A typical demodulator NC-OFDM is shows in Figure 1.7(b): each sub-carrier at the input is represented by each input of the FFT block. The main difference from the modulator scheme is represented by the pruning block: knowing which subcarriers are disabled, in order to reduce the computational work of the block FFT, the redundant operation inside the FFT block can be eliminated, in correspondence at the zero inputs [11].

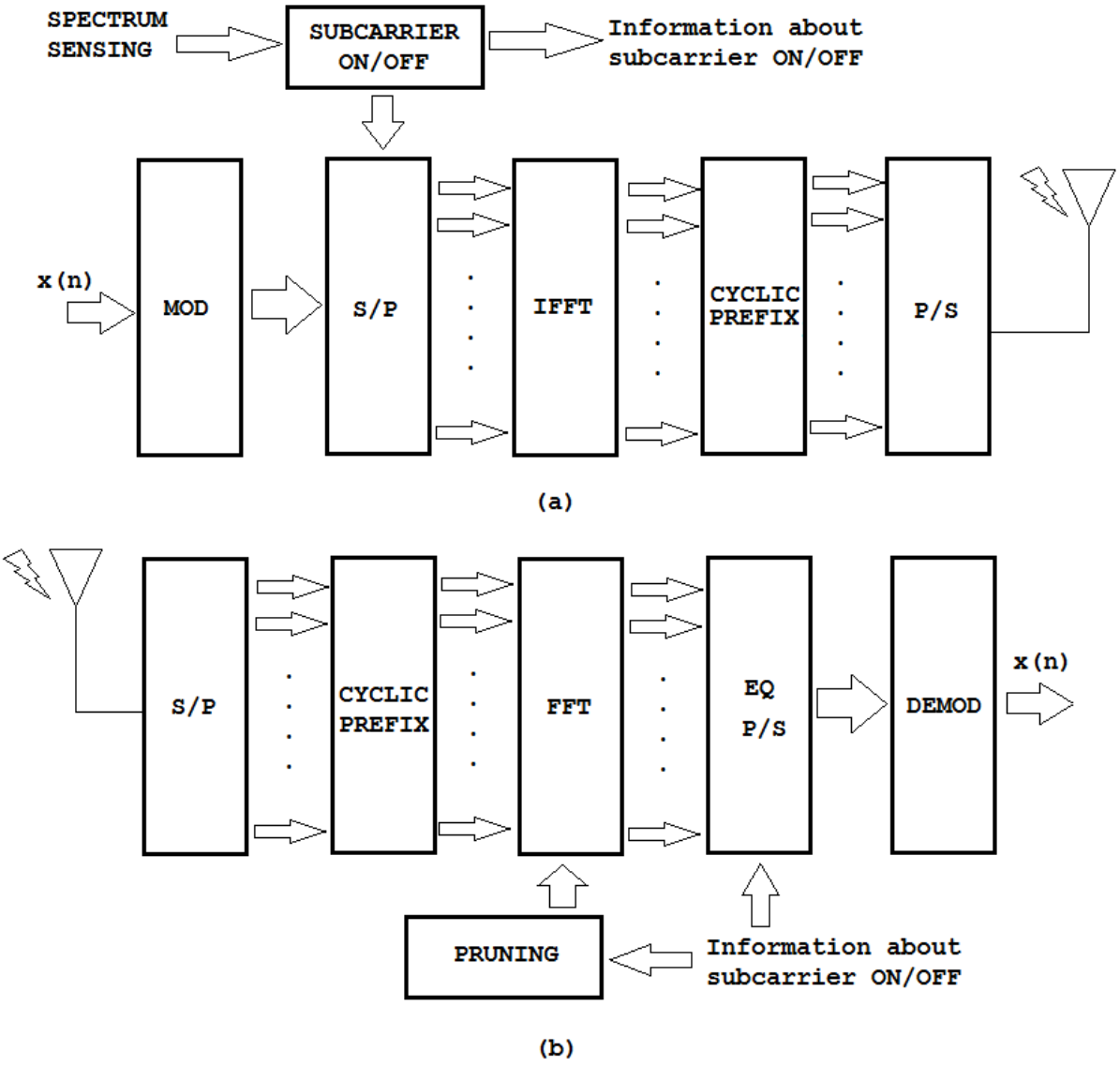


Figure 1.7: (a) NC-OFDM transmitter; (b) NC-OFDM receiver [23].

Therefore, a NC-OFDM system provides the necessary degree of spectral agility required by Cognitive Radio systems. In a CR system, the bandwidth of a user is not always permanent, and the spectrum is shared by several users. NC-OFDM modulation gives the opportunity for secondary users to turn off portions of its range, in conjunction with the primary users transmission. In [12] is presented the NC-OFDM technology based on a block diagram of a typical architecture CR.

1.4 Fast Fourier Transform

Any OFDM radio receiver is characterized by Discrete Fourier Transform block. In cognitive radio applications and in many radio applications, adopting high performance DFT algorithms which allow the calculation of the DFT with a lower computational complexity, results very important. Therefore, in the last years, different algorithms called algorithms FFT (Fast Fourier Transform)[13] have been developed among which, one of the main is called Radix-2 FFT that we will introduce in this chapter.
This discussion begins from the definition of DFT, by posing as final objective to make the most efficient possible calculation of the vector X [k] from a vector of input samples x [n].

A DFT and IDFT process can be mathematically defined by the equations (3) and (4):

$$(DFT) \rightarrow \qquad X_q = \sum_{k=0}^{N-1} x_k \; e^{-j(2\pi \cdot kq/N)} \qquad \text{where} \qquad (\, q = 0,1,..,N-1\,) \tag{3}$$

$$(IDFT) \rightarrow \qquad x_k = \frac{1}{N} \cdot \sum_{q=0}^{N-1} X_q \; e^{j\left(2\pi \cdot \frac{kq}{N}\right)} \qquad (\, k = 0,1,..,N-1\,) \tag{4}$$

In order to calculate the generic X_q sample transformed, N multiplications and $N-1$ sums are required; since the total samples that have to be calculated are N, the total number of multiplications and additions that a normal DFT algorithm has to do are N^2 and $N(N-1)$. Typically, in terms of computational cost the operations of $N(N-1)$ sums do not present major problems with respect to the problems that arise in making N^2 multiplications, in general, between complex samples.

Thus, it is possible to reduce the number of multiplications, using some properties of the transform. From the definition of DFT, the term $e^{-j(2\pi/N)}$ depends only on the number of points of our DFT. Then, it is considered as a constant term, calls $W_N^i = e^{-j(2\pi/N)i}$ = "*Twiddle Factor*". This coefficient has two very important properties called:

1) Periodicity properties: $W_N^{k+N} = W_N^k$
2) Symmetry properties $W_N^{k+N/2} = -W_N^k$

Due to these properties, the algorithms defined as FFT algorithms can be presented.
The most popular algorithms classified as FFT is the Cooley-Tukey algorithm [10] that consists into the split of the N-DFT computation in two different computational operations through two DFT, of which, one with N_1 points and one with N_2 points. This algorithm has the advantage that it can be applied recursively to the calculation of any N points DFT with $N = 2^n$: at each iteration, each $\frac{N}{2^l}$ points DFT is decomposed into two $\frac{N}{2^{l+1}}$ points DFT.
Radix-2 FFT can be classified in two types: *decimation in time* and *decimation in frequency.*

From the definition of DFT, further developing shows:

$$X_q = \sum_{k=0}^{N-1} x_k \, W^{k \cdot q} = \sum_{k=0}^{\frac{N}{2}-1} x_k \, W^{k \cdot q} + \sum_{k=\frac{N}{2}}^{N-1} x_k \, W^{k \cdot q} \quad (5)$$

Imposing $z = k - \frac{N}{2}$:

$$X_q = \sum_{k=0}^{N/2-1} x_k \, W^{k \cdot q} + \sum_{z=0}^{N/2-1} x_{z+N/2} \, W^{(z+\frac{N}{2}) \cdot q} =$$

$$= \sum_{k=0}^{N/2-1} x_k \; W^{\,k\cdot q} \; + W^{\frac{N}{2}\cdot q} \sum_{z=0}^{N/2-1} x_{z+N/2} \; W^{\,z\cdot q} \qquad (6)$$

From the property $\mathrm{W}^{\frac{\mathrm{N}}{2}\cdot\mathrm{q}} = \mathrm{e}^{-\mathrm{j}\,\pi\cdot\mathrm{q}} = (-1)^{\mathrm{q}}$:

$$X_q \; = \; \sum_{k=0}^{N/2-1} x_k \; W^{\,k\cdot q} \; + (-1)^q \sum_{z=0}^{N/2-1} x_{z+N/2} \; W^{\,z\cdot q}$$

Now, there will be two cases from the separation of the N samples sequence, in two sequence of even and odd temporal position:

$$q\ even\colon \rightarrow \qquad X_{2q} \; = \; \sum_{k=0}^{N/2-1} (x_k + x_{k+N/2})\, W_{N/2}^{pk} \qquad (7)$$

$$X_q \; = \; \sum_{k=0}^{N-1} x_k \; W^{\,k\cdot q} \; =$$

$$q\ odd\colon \rightarrow X_{2q+1} \; = \; \sum_{k=0}^{N/2-1} (x_k - x_{k+N/2})\, W_N^k \, W_{N/2}^{pk} \qquad (8)$$

Hence, we traced to what has been said previously, with two summations representing two N / 2 points DFT respectively of the sequence of odd and even samples input. Each DFT of length N / 2 require $\frac{N^2}{2}$ number of multiplications.

Repeating this procedure and breaking the two summations in as many summations always obtained by separating the odd and even terms, Radix-N algorithm can be obtained.

Stopping the DFT calculation to a only two samples sequence, it is possible to show that the number of iterations that must be accomplished (also called stage) is defined as $\log_2(\mathrm{N})$. This discussion assumes that the initial number of samples is a power of two, in particular $\mathrm{N} = 2^{\mathrm{m}}$.

The initial stage, which provides for the calculation of two points FFT, takes the name of "butterfly", whose fundamental operations are multiplications and sums: as shown by Figure 1.8, a butterfly takes as input two values (in general complex), performs a multiplication by a coefficient (Twiddle Factor) and then, performs a sum and a difference:

$$X_0 = x_0 + x_1; \qquad X_1 = x_0 - x_1;$$

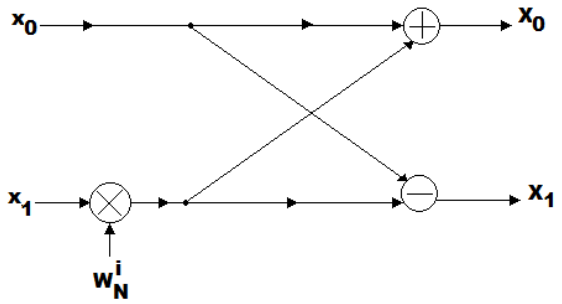


Figure 1.8: Single butterfly scheme.

The computational cost of the entire algorithm, it is profoundly decreased. Considering a sequence of $N = 2^m$ sample in input at every stage of this architecture, $\frac{N}{2}$ butterfly have to be calculate each of which, it must compute one multiplication. The numbers of stages are $\log_2(N)$ and therefore the total number of multiplications required calculating the entire algorithm will be $\frac{N}{2}\log_2(N)$ and $N\log_2(N)$ complex ads. Thus, the gain with respect to the number of complex multiplications, compared to the case of simple DFT is improved.

This gain depends by N points FFT and observing (9) it will be better with high number of N input samples.

Figure 1.9 shows the complete scheme with full radix-2 decimation-in-time according to the algorithm just presented. Each stage will calculate N/2 single butterfly, and there are 4 stages [14].

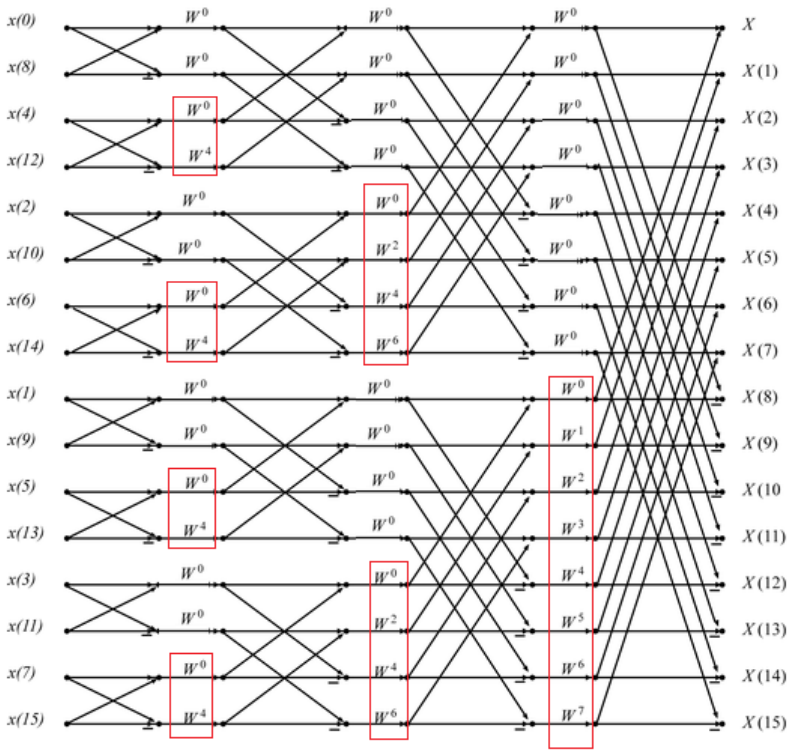


Figure 1.9: FFT Radix-2 scheme N=16 [14].

To complete the presentation of the algorithm, before starting with the calculation of the FFT, bit reverse operation has to performed. Considering the binary representation of the vectors indicated and performing the swap of bits having a symmetrical position with respect to the center of the carrier, the bit reverse operation can be obtained. This is also called Bit Reverse Order.

By studying this calculation algorithm FFT Radix 2 it is understood that to reduce the complexity of an FFT algorithm, and then to reduce the number of total multiplications, it is necessary to operate with butterfly structures, but it isn't the only solution that can be adopted to this end.

In particular, supported on the same theoretical concepts and on the basis of the symmetry of the twiddle factors coefficients, the same algorithm can be extended when the length of the sequence input samples (N) is a power of four: this is known as FFT Radix-4 algorithm [15].

As an algorithm for Radix-2, also an algorithm Radix-4 is based on the concept of butterfly, the basic element that makes up the core of the FFT and from which, through simple iterations, it is possible to schematize any N points – FFT.

In the Radix-4, the input vector to be transformed is divided into 4 vectors: of each of them we can calculate the FFT. As reported in [16], an algorithm Radix-4 has advantages over a Radix-2 algorithm, since the number of additions to be made remains constant, while the number of multiplications required decreases by 25% (in fact, if each butterfly radix-2 require 1 multiplication, each butterfly of a radix-4 algorithm require only 3 multiplications. Therefore, since a butterfly radix-4 is equivalent to four radix-2 butterfly, using a solution in radix 2 we will have four total multiplications, while with the radix-4 we will have only three of it). This advantage is more evident for large N-FFT point. However in [16], it is emphasized as an algorithm Radix-4 presents a much greater architectural complexity of an algorithm Radix-2; for this reason, in case it is explicitly requested algorithm to operate with very long sequences (N>>2048), the Radix-4 is not often used, because the gain obtained by saving 25% of the multiplications is lost because of greater architectural complexity, which means increase of area, power consumption and cost of whole architecture.

CHAPTER 2

State of the art of FFT pruning

A transmitter block NC-OFDM Cognitive Radio has the ability to cancel subcarriers not required for communication, at certain times and in certain point's time of the spectrum.
Because of most parts of the spectrum are unused by the transmissions of primary users then the secondary users (users without license of transmit) could theoretically occupy this portions of band. In a similar mode, they provide to interrupt their communications at the portions of spectrum occupied by primary users (users with license) thus avoiding any kind of interference.
As a result of this operation, a Cognitive Radio receiver might operate with long sequences of input equal to zeros, caused by long zero portions of spectrum. One of the main parts of a radio receiver is constituted by an the FFT block, which picks up the incoming samples from the channel, and, through a Fourier Transform operation, enables the recognition and reconstruction of subcarriers. FFT block can be realized in various ways, in order to obtain greater energy efficiency and to decrease the time of calculation, for example, adopting solutions as Radix-2 FFT.
However, in correspondence of samples sequences characterized by long sub-sequences of zeros standard FFT algorithms are computationally inefficient due to the operations of sum and multiplication performed unnecessarily on null values.

The first hints about the idea of FFT pruning algorithm was introduced in [2]; in this article a pruning algorithm is presented aimed at eliminating the unnecessary operations

for the calculation of the outputs in the presence of zeros input. In [19] Markel was able to demonstrate the importance in terms of computation reduction that an application of this type involved. From that point on, different algorithms have been proposed in the literature, calls as "pruning", in order to reduce the number of multiplications and additions performed in correspondence of the zeros input.

In [17] the difference in terms of computational complexity among several algorithms FFT pruning was analyzed. It is demonstrated that a radix FFT algorithm Split-Pruning has a better computational complexity than the other. Our implementation starts right from the result of this work, and then it is inspired to a similar type of algorithm but completely redefined.

However, as presented in [18], many of these algorithms don't have efficient architectures for realization on FPGA, due to the arbitrary distributions of zero inputs that makes difficult hardware implementation.

The state of the art in the face of this important issue has seen the development of dynamically reconfigurable FFT algorithms, called DPR, (Dynamically Partial Reconfigurable) based on the concept of inputs IFFT pruning.

In [19] the authors have implemented a pruning algorithm, to eliminate unnecessary operations of multiplication and addition, constructs "if" and "else" are used on the elements of a pruning matrix. The disadvantage of this approach is that the execution time of these instructions is too high, and furthermore, their use is not efficient for FPGA implementations, hence it reduces the advantage of having a reduced complexity of the algorithm.

Through the pruning matrix is possible to map the distribution of the zero input to the FFT block. In [20], an algorithm that uses a matrix $N \times r$, where each element of the array corresponds to a particular input at each stage of the FFT is proposed. This matrix requires enough configuration time, and an important space in memory. Each column of the matrix corresponds to each stage of the FFT, while in each column N values can be finding, referred of the N outputs of the previous stage. Checking the value of the corresponding elements of the array, we can understand which of them should be calculated or not.

In [23] a low-power 2048-FFT design implemented through a radix-2^4 for NC-OFDM in Cognitive radio is presented. The authors have developed a different design based on an efficient zero flag generation technique and they obtained an improvement of power consumption of RAMs up to 22.3% when the number of zero inputs increases. They have not reported the power results of entire FFT architecture and their approach doesn't use a declaration of pruning matrix in order to improve the performances of FFT.
A similar solution is presented in [11]: an energy-efficient Fast Fourier Transform algorithm for cognitive radio is implemented on multi-processor architecture.

The objective of this thesis is to reduce the size of the matrix configuration and the hardware complexity of the FFT block, and reduce the power consumption of the device in case it has to operate with long sequences of zeros input.
To achieve this purpose, it was first developed a Radix-2 FFT on FPGA, in order to minimize the hardware complexity. Hence, through a pruning matrix it is implemented the pruning, through a operation of deactivation of the multiplication and addition operations by clock-gating technique.
This work also provides a complete synthesizable VHDL code, which describes the whole architecture FFT and the implementation of pruning. Because of the decision to use a VHDL code to define it, we can create a project without having to first select a device on which to implement it. Moreover, because of the characteristic of portability of the code, it can be reused in other projects on other types of devices.
This proposal does not take the objective of reducing the calculation time of the algorithm but it would like to improve the power consumption only.

CHAPTER 3

Architecture of FFT pruning

In this chapter, the architecture of a FFT pruning system developed is presented. In the first part, the operation of the FFT core system where the pruning part will be implemented, is briefly presented. In the remaining part of this chapter, a more detailed description of the pruning implementation is introduced, from the motivation on the base to the techniques used to run it.

3.1 General block scheme

As shown by Figure 3.1, the general architecture FFT is composed of:

- 8 *RAM memory* banks (Read Access Memory) for storing input/output samples of FFT algorithm;
- 1 *ROM memory* bank (Read Only Memory) containing the Twiddle Factors coefficients;
- 2 *Interconnect* blocks responsible of routing data contained in the memory blocks;
- 2 *Butterfly* blocks each containing the basic architecture of one butterfly;
- 1 *Control Unit* block is the heart of the FFT architecture; it is responsible of logic and management of all control and command signals;
- 1 *Address Generation* block, responsible for generating the address to read/ save the data from/to memory;

- 1 *Pruning Matrix Memory* bank to store the *pruning matrix*;
- *1 Control Pruning Unit which will perform the job of interpreting the pruning matrix contained in the memory, and then to manage appropriate enabling signals.*

3.2 FFT core architecture

The design of FFT architecture depends on the type of algorithm adopted. The main FFT algorithm used in the field of the telecommunication is the FFT Radix-2. *N* points FFT is based on a different iterations (called stages $n = \log_2 N$) supported by a symmetrical algorithm's logic. The most important element of this algorithm is the butterfly.

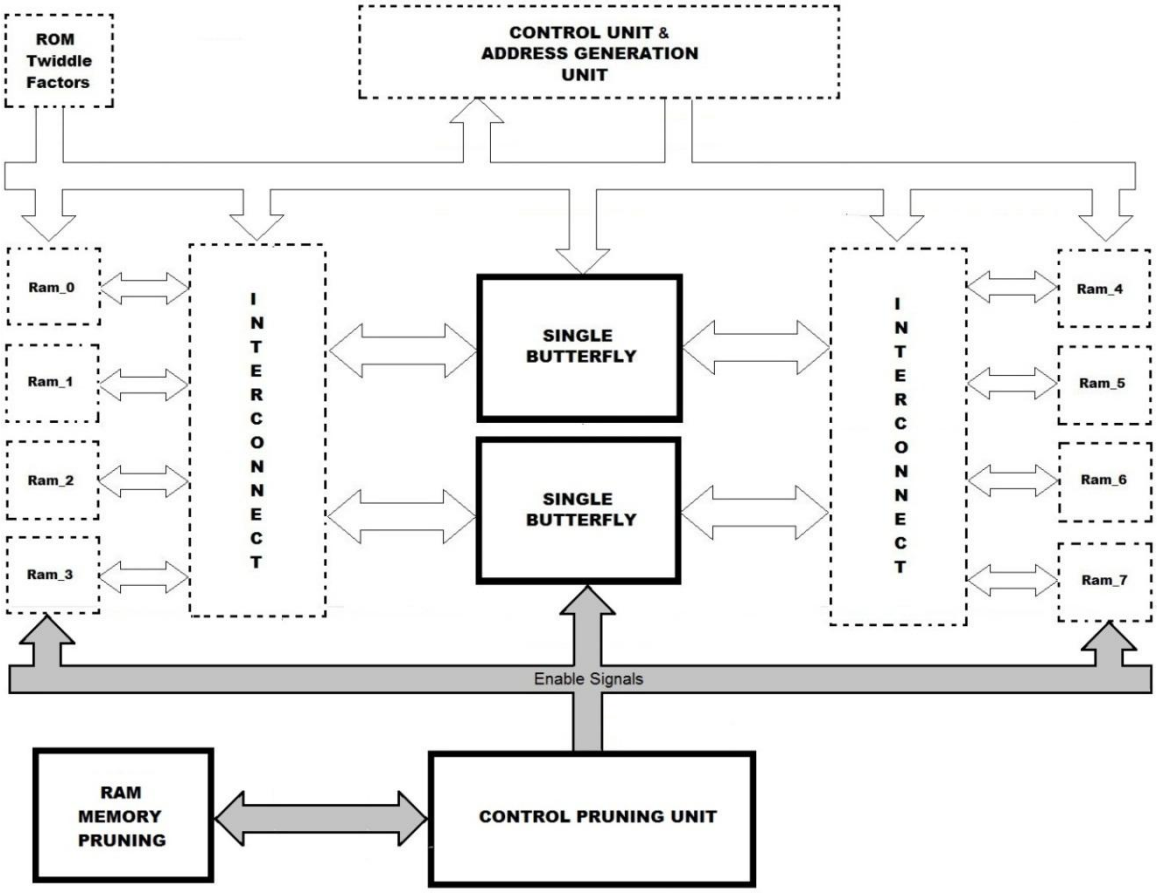


Figure 3.1: FFT pruning block scheme.

As described in Figure 3.1, the architecture is composed of two sets of memories, called *Set_A* and *Set_B* each consisting of four banks of RAM. The vector of N input samples is re-ordered according to the algorithm of reverse order, and it is divided into two parts called First Half and Second Half. The algorithm stores the samples of First Half in the Ram_0 and Ram_1 memory banks alternately, while the samples of the *Second Half* will be stored in the *Ram_2* and *Ram_3*.

Figure 3.2(b) shows an example of storing of a 16 points FFT algorithm at the first stage, and the same concept can be extended to any FFT size. Now, a signal "*start*" (managed by the Control Unit), starts the algorithm. The block *Address Generation* through an address signal called *read_addr* (defined by a binary counter to $\log_2 N/4$ bits) decides the memory location from the samples took. The samples coming from *the Ram_0* and *Ram_1* are directed towards the first butterfly, while samples from the memory banks *Ram_2* and *Ram_3* are directed towards the second butterfly as shown in Figure 3.2 (c). After 3 clock cycles, they will begin to flow into the butterfly.

As presented in [16] in each butterfly, it performs 16-bit operations of multiplication and addition. The time required by each butterfly to perform operations on each sample and validate the respective outputs is of 4 clock cycles.

The results of each butterfly are routed to the memory banks of *Set_B*. The address to store the individual samples inside each memory bank are produced by *Address Generation Unit*. A signal *Store_add* identifies the memory location of the destination for the results of the upper branch of each butterfly while *Store_Sub* identifies the memory location for the lower one. The memory bank that contain the result is selected by bit 0 of the signal *read_addr*. If *read_addr(0)= '0'* then the results of the first and second branch of the butterfly_0 are stored in *Ram_4* and *Ram_5* respectively , while the results of the *butterfly_1* are stored in *Ram_6* and *Ram_7* as in Figure 3.2(c). If *read_addr (0) = '1'* the results of the first branch of the *butterfly_0* are stored in *Ram_5* while those of the second branch are stored in *Ram_4*. Likewise for the second butterfly a similar procedure is defined. At the end of each stage of the Radix-2 FFT algorithm shown in the Figure 1.9, the results of the stage are present in one of the two sets of memory banks. In the next stage the algorithm restarts and all of these operations are the same but in the opposite direction (*Set_B -> Set_A*).

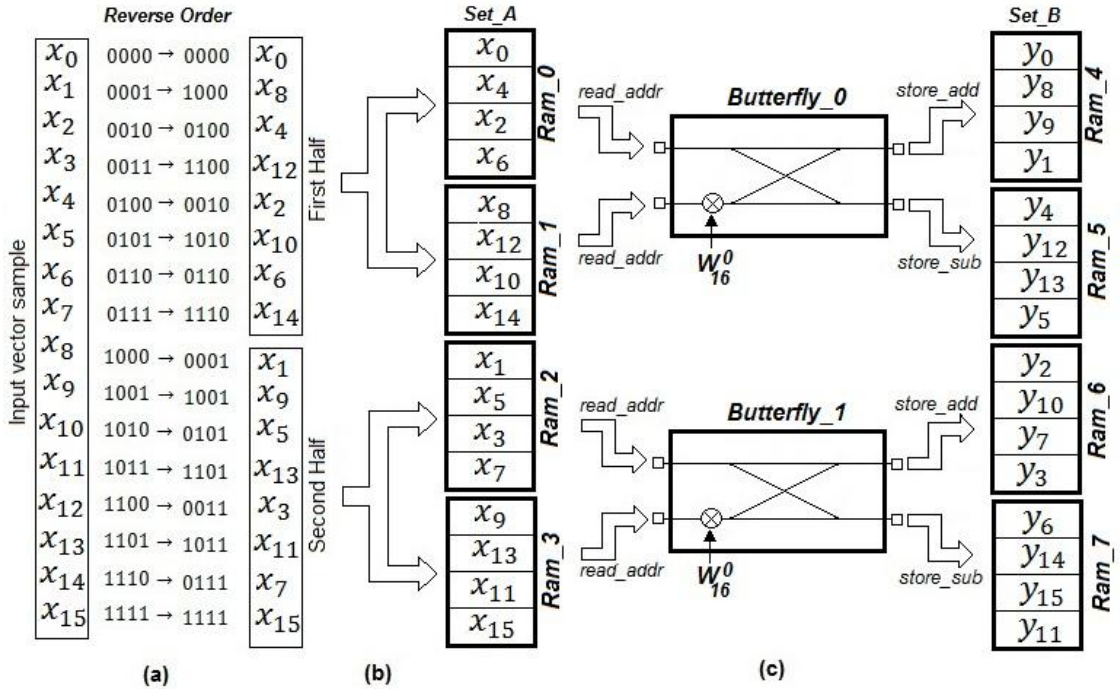


Figure 3.2: scheme of read and store sample.

A sample required nine clock cycles for an execution of an entire passage between two sets of memory. Defined this procedure it is possible to determine in advance the computation time of the entire algorithm: denoting by *n* the number of stages and *N* the number of points of FFT, each stage of a Radix-2 is composed of *N/2* butterflies. Having two blocks for the calculation of two butterflies simultaneously, only the computation time for N/4 butterflies per stage can be considered. Hence, the number of clock cycles to perform an entire FFT algorithm with *N* points is *9 + [2*(N/4)]*n*.

3.3 Objective of pruning

Due to the complexity savings, a FFT pruning architecture based on the particular FFT Radix-2 (decimation-in-time) is developed, to which has been added a further logic for the implementation of pruning.

FFT architecture is characterized by two parameters that represent the quality of the system: the simulation time and power dissipation. These two parameters are linked together by the relation $E = P \times t$ that means both are fundamental for decreasing the energy consumption of the device. The pruning algorithm is based on the target to reduce the power consumption of the architecture when there are zero values in input to the FFT. The presence of zero samples as input of an FFT is a very common situation in the field of Cognitive Radio, or for DSA. This capacity to adapt has the advantage to exploit entire portions of spectrum otherwise left unused, but has the problem of not ensuring continuous transmissions producing many empty intervals transmission.

After presenting the FFT algorithm (in general way), the core of this work, as implementation of a pruning structure, is introduced. As presented in Figure 3.1, the pruning part is represented by two additional blocks that are added at the base FFT architecture without changing its behavior. In particular, we have chosen (for simplicity of realization), to adopt the timings sets by the FFT core keeping the execution time constant. This decision is supported also because the symbol timing is constant.

In fact, another way to perform an algorithm pruning is to improve both the execution time and the power dissipated, in order to have a more significant decrease of the total energy used. However, the initial design phase that led us to define the algorithm FFT, required a lot of energy and effort, especially in the phase of synchronization between Address Generation Unit, Control Unit and pipeline samples inside of each butterfly. For this reason, for the implementation of pruning, we have chosen to not modify the time structure of our architecture, just trying to improve the power dissipated by individual blocks.

The most important timing signals are the *start* signals, *begin_stage* and *begin_pruning*. The *start* signal (active high) indicates the beginning of the FFT algorithm, the signal *begin_stage* (active high only for one clock cycle) indicates the beginning of each stage of the cipher Radix-2 FFT, while the signal *begin pruning* (pulse of a clock cycle active high) indicates the beginning of the phase of pruning. The pulses *begin_stage* will be equidistant at a number of clocks that depends exclusively on the number of points of the FFT. This signal is activated whenever the last result coming out of the butterfly is stored in their memory bank. In sync with this signal, the *begin_pruning* signal is set, which will make starting the pruning mechanism that is presented in more detail in Chapter 3.

3.4 Presentation of butterflies blocks

The main blocks that we modify in order to reduce the power dissipated are the two butterflies blocks. These two blocks were presented in [16] and are characterized by the architecture presented in Figure 3.3. Before starting the description of their content the sample format is briefly introduced.

In the signals analysis, in general for sinusoidal signal, a generic sample X_0 is represented by a complex number:

$$X_0 = X_R + jX_i \tag{10}$$

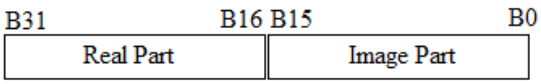
That uniquely identifies the phase and amplitude. In particular, observing these two parts the amplitude of the generic sample can be known, simply calculating the module of the complex number:

$$|X_0| = \sqrt{{X_R}^2 + {X_i}^2} \tag{11}$$

The calculation of the phase is made possible by:

$$arg(X_0) = arctg\frac{X_i}{X_R} \tag{12}$$

Thus, a complex signal can be regarded as the combination of two real signals: in this thesis, the samples from the channel are being composed of two parts (real and imaginary part) each represented by 16 bits:



In the case that one of the two parts (real or imaginary) is in negative form, its representation is done through the form in *two's complement*. For the theory of the two's complement, it is necessary write the number in unsigned binary representation, then individual bits have to be inverted and 1 has to be added to the value of the number found. In input to the butterfly, we have two samples and a Twiddle Factor coefficient required for internal operations. From a mathematical point of view, three types of steps in each butterfly are occurred: two sums and a complex multiplication by a given coefficient. In fact, each element in the input of a butterfly is generally a complex number, and can be decomposed into a real and an imaginary parts, each represented by 16-bit.
Each element in the input of butterfly is generally defined as complex number, and can be decomposed into a real part and an imaginary part:

Sample: $$S = S_{Re} + j\, S_{img} \tag{13}$$

Twiddle Factor: $$W = W_{Re} + j\, W_{img} \tag{14}$$

The main part in the calculation of a butterfly is the multiplication between a Twiddle Factor and a sample input. We can explain it as:

$$W \cdot S = (W_{Re} + j\, W_{img}) \cdot (S_{Re} + j\, S_{img}) =$$
$$= W_{Re}S_{Re} + jW_{Re}S_{img} + jS_{Re}W_{Img} - S_{img}W_{Img} \tag{15}$$

Adding and subtracting the terms $S_{Re}W_{Img}$ e $S_{Img}W_{img}$:

$$= W_{Re}S_{Re} + jW_{Re}S_{img} + jS_{Re}W_{Img} - S_{img}W_{Img} \pm \boldsymbol{S_{Re}W_{Img}} \pm \boldsymbol{S_{Img}W_{img}} =$$
$$= W_{Img}(S_{Re} - S_{img}) + S_{Re}(W_{Re} - W_{img}) + j\left[W_{Img}(S_{Re} - S_{img}) + S_{Img}(W_{Re} + W_{img})\right] \tag{16}$$

With this representation, to make a multiplication between two complex numbers *three multiplications and five additions are needed*. In order to do these operations, for the architecture defined in [16], the number of arithmetic blocks required, can be reduced by exploiting a particular implementation: this implementation provides to divide the sample between their real and imaginary part and split the calculation in two clock cycles as shown on by Figure 3.3.

In a first cycle of clock, the operations are:

$$A = W_{Re}S_{Re} \quad and \quad B = W_{Re}S_{img} \tag{17}$$

In a second clock cycle we will calculate:

$$C = S_{Re}W_{img}\ and \quad D = W_{img}S_{img} \tag{18}$$

At the end of the C, D, A, B calculations, the operations of addition are performed:

$$\boldsymbol{W \cdot S} = (\boldsymbol{A - D}) + \boldsymbol{j}(\boldsymbol{B + C}) \tag{19}$$

Therefore, with just two multipliers and two adder blocks used in 2 clock cycles, the multiplication between two complex samples of 32 bits each can be realized.

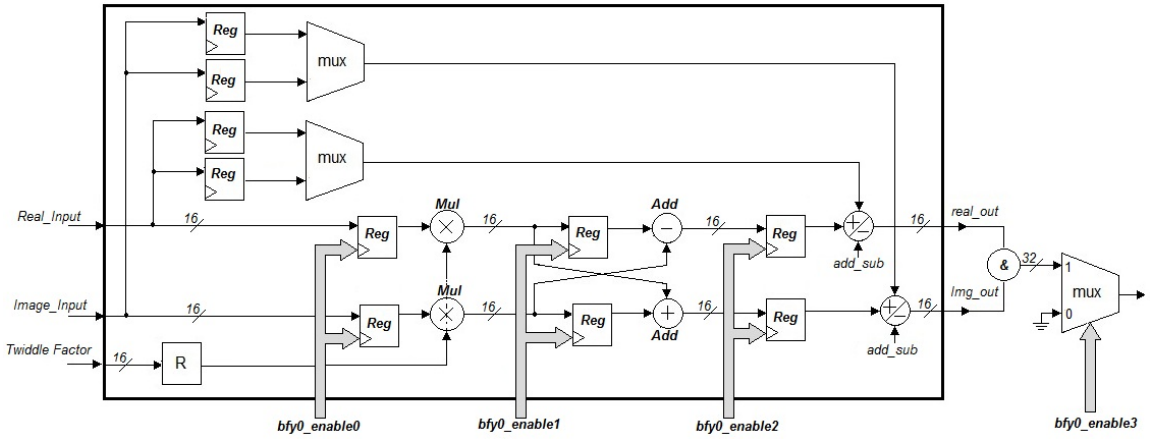


Figure 3.3: Butterfly internal scheme block [16].

However, if we consider N-points FFT with *N =2048,* there are 11 stages each consisting of 1024 butterflies. The total number of butterflies is 11264 and then, the total number of multiplications is $11264 \cdot 4 = 45056$. In the same way, the total number of sums (including those required by the product between two complex numbers) is $(11264 \cdot 2) + (11264 \cdot 2) = 45056$.
Each of these operations requires a certain amount of energy to be completed, especially the multiplications among complex numbers.

The binary multiplication between numbers with more bits represents the more complex arithmetic operation that our device has to run: to perform this, the FPGAs can implement various techniques, through use of different logical structures: one of these is the *scaling multiplier accumulator*: this kind of technique multiplies the first bit on the right of the *multiplier* with each bit of the *multiplying*. Then, it proceeds in the same way with the second bit of the *multiplier*, reporting the result of the product in the line below starting from a leftmost position, and so on until the last bit of the multiplicand. The sum of the partial products determines the product of the binary multiplication. Other techniques can be used in modern FPGAs of today; in general, based on our example in Figure 3.4, in our algorithm the multiplication between 16-bits samples has to be done then, for each multiplication the calculator should work in expensive way.

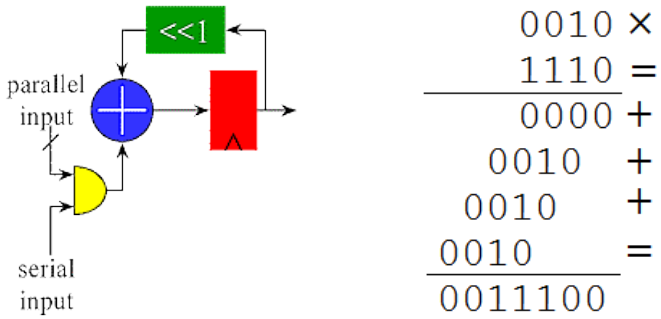


Figure 3.4: Example of scaling multiplier accumulator between two 4-bits number[21].

However, if the inputs to an arithmetic block are two digits zero, regardless of the length of them, the calculation requires no operation because the result will always be zero.

Hence, the idea of pruning is to disable all arithmetic blocks located inside of the butterfly, when we have zero values at their input.

The technique used to disable these operations is the *clock gating technique*. This technique provides to stop of arithmetic blocks and, in general, of all those blocks that do not work, by disabling the clock signal for a certain time period. For this reason, as shown in Figure 3.3, we have modified the butterflies blocks defined in [16] in order to condition the functioning of the arithmetic blocks, by introducing three enable signals: *bfy0_enable0*, *bfy0_enable1* and *bfy0_enable2*. In addition, a further signal *bfy0_enable3* at the output of the butterfly allows to put the output equal to zero.

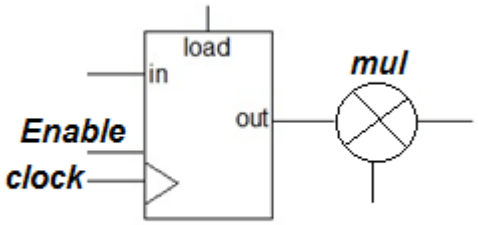


Figure 3.5: Example of Clock Gating Technique.

The architecture of each butterfly block has three pairs of registers, each inserted upstream of each arithmetic block. Based on VHDL code, the registers behave like blocks of delay of one clock cycle for each output result from arithmetic operations. When the clock pulse arrives, each register transfers its input to the output, and a multiplier (or an adder) carries out its operation. Introducing an enable signal in the registers present before each arithmetic blocks, if the signal *enable is '0'* then the clock in the register is disabled and its output does not change. Figure 3.6 shows the RTL viewer diagram obtained with Quartus II. It displays the Analysis & Elaboration results for a generic VHDL designs, provides its gate-level scheme.

We can see the internal enable signals to the butterfly and the presence of register blocks before of multipliers and adders blocks.

The green block of the Figure 3.6 presents the internal structure of the butterfly specific for the multiplication between the Twiddle Factor and the our input sample. Inside it, there are two multipliers and two adder's blocks, in order to realize the operations managers by equation (19).

The algorithm implemented to generate the enable signals is presented in the next chapter, with the motivation of how our architecture was done in VHDL.

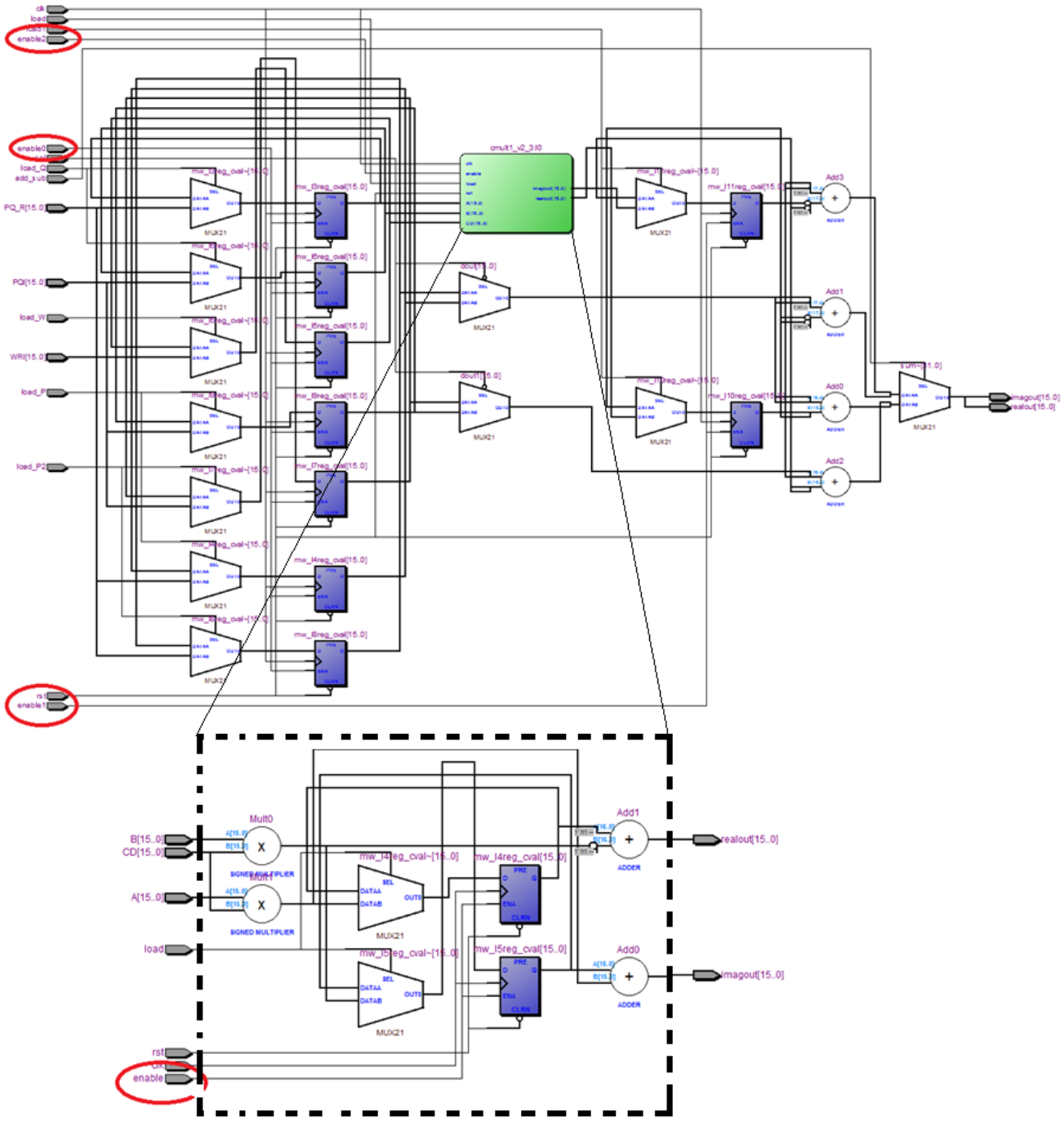


Figure 3.6: RTL View of internal butterfly.

3.5 Theory of pruning matrix

As described in the previous paragraph, each enable signal generated by the Pruning Control Unit, brings the information related to the status of a particular butterfly calculated by the algorithm. The generation of these signals is directly dependent on how much information on the state of the FFT inputs are known and the technique to map and interpret them. In the example shown in Figure 3.7, a Radix-2 16-point FFT scheme (adopted by our core) is considerated. It is referred to certain distribution of zeros in input (Stage0). At each stage of the algorithm, N / 2 butterflies are calculated some with both zeros input. This butterfly could be eliminated by algorithm calculation, simply setting their outputs equal zero.

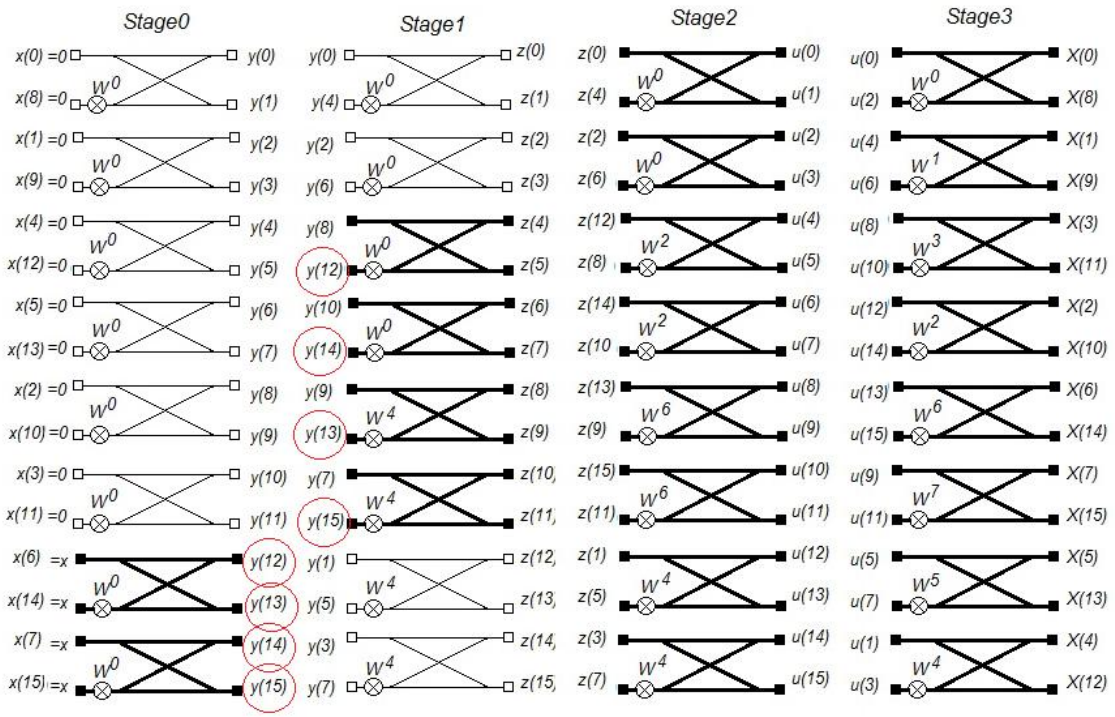


Figure 3.7 Schematic Radix-2 algorithm for N = 16.

In addition, the presence of zeros input in the first stage is propagating along the structure of the algorithm and in the second stage (stage 1) and other butterflies can be eliminated.

An example of 16-point FFT is reported in Figure 3.7 : using only 16-points FFT, the great effect of the zeros propagation along the stages is not demonstrated with sufficient resolution.

In literature the idea of performing the pruning based on input samples state, has already been addressed in several ways, one of which was introduced by Rabjanksi [22]. His theory is based on the construction of a $\mathrm{N} \times \log_2 N$ matrix, where the elements of each column represent the state of the inputs to each stage. This matrix was introduced in order to propose a mathematical solution to the problem of finding the zeros, but without considering the real construction of it, and its hardware implementation.

The proposed solution uses the same theoretical principle expressed by Rabjanksi, but in slightly different ways: in order to implement this architecture in FPGA hardware, the problem of complexity of a system and with the resources used by the system has to be analyzed. Pruning matrix proposed is a $\mathrm{N}/2 \times \log_2 N$, because it does not refer to the state of the individual inputs, but it refers to the state of a single butterfly. For this reason, its size will be reduced by 50% factor:

$$M = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \qquad \text{Rabjanksi's matrix} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

This matrix has to be stored in a memory and it has to be generated by an algorithm. Hence, the reduction of the matrix size translates into a reduction of hardware resources and in the reduction of matrix construction time. This last factor is particularly important in the design phase, because our content requires that this matrix has to be set dynamically

based on the channel conditions. For this reason, the construction of such a matrix cannot take excessive generation time. The next paragraph reports the comparison (demonstrative purposes only) between the two types of matrix for the 16-point FFT structure of Figure 3.7.

3.6 Construction of the Pruning Matrix

The Pruning Matrix is a binary matrix composed of $N/2$ rows and $\log_2 N$ columns. Each bit of each column refers to a particular butterfly of the algorithm proposed and the order of the bits in each column has to correspond to temporal order where the algorithm calculates the butterflies.

If both inputs of a butterfly are zero the corresponding bits in the array will be set to zero. Conversely, if at least one of the two inputs is different from zero, thus the butterfly cannot be excluded from the calculation, and thus the corresponding bit in the array is set to one. This type of pruning is generically defined as partial pruning because it is used only if both input of a butterfly are zeros, without recognizing these butterflies with only one zero input.

In the previous step, the problem of the generation of the matrix has to be solved. Figure 3.8 shows the flowchart implemented to load the contents in the memories of architecture.

If both butterfly inputs are equal to zero, the corresponding bits in the matrix will be set to zero. Conversely, if at least one of the two inputs will be different from zero, hence the butterfly cannot be excluded from the calculation, and the corresponding bits in the array will be set to one. This type of pruning is generically defined as partial pruning because we don't consider the single input of a butterfly when we want to decide about its deleting. The construction of the pruning matrix was performed through a C language program called *make_matrix_pruning.c*; starting from observation of the samples stored in RAM memory

banks of our architecture, this algorithm receives as input the value of the three variables:

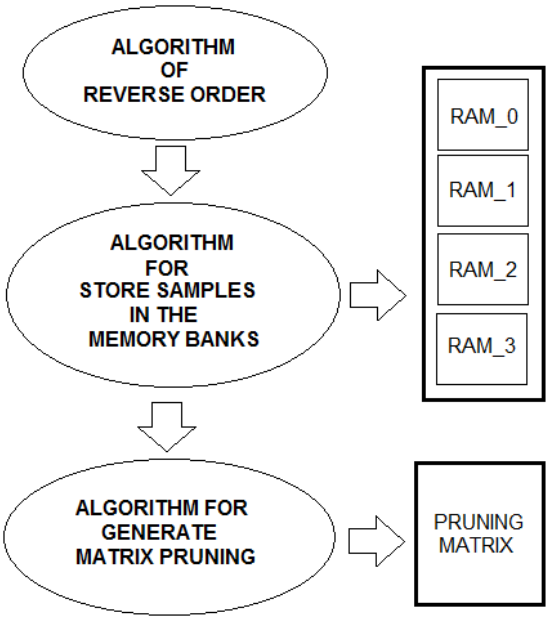


Figure 3.8 Flowchart to store of samples.

#define N	*2048*
#define NUM_STAGES	*11*
#define THRESHOLD	*0*

From these variables, the number of inputs N and the number of stages of FFT con be set with the THRESHOLD parameter the threshold below which the samples considered null and void can be chosen. In a real application, since all samples are always subject to channel noise that will lead them to have low values (but still non-zero) then, the definition of a THRESHOLD is necessary.

The first phase of this algorithm defines the first column of the memory of pruning, through a simple conditional construct:

$$IF\ (data1 \leq THRESHOLD\ \ \&\&\ \ data2 \leq THRESHOLD)\{$$
$$A[j] = 0;$$
$$else$$
$$A[j] = 1;$$

This construct will be used two times: at the first time the terms *date1* and *date2* indicate the contents of memory and *RAM0, RAM1* while ay the second time they indicate *RAM2* and *RAM_3* memory contents. In this way, a vector *A [N / 2]* is constructed. This vector contains the first column of the matrix. Instead, the construction of the remaining columns of the matrix depends on the type of FFT algorithm is implemented, but we do not take care of it in this thesis. Each column of the matrix is built through the observation of the previous and through to the knowledge of the FFT algorithm.

CHAPTER 4

Development on FPGA

In this chapter, the implementation of the FFT pruning architecture is presented. In the first part, a brief presentation about the Field-Programmable Gate Arrays (FPGA) technologies is inserted, highlighting the advantages and disadvantages of the use of this technology. Hence, in the final part of this chapter, the statements of the architecture entities responsible for the pruning are described.

4.1 Briefing about FPGA technologies

An FPGA is a device consisting of a large number of programmable logic components said Logic Elements (LE), I/O blocks and storage elements called flip-flip, which are connected to each other through a network of programmable interconnects. [24] Therefore, an FPGA can be represented as an array of identical logic cells that form the basis for the construction of any circuit also very complex. Through the configuration of the LE is possible to perform sample logic gates (i.e. AND, OR) but also complex combinatorial functions.

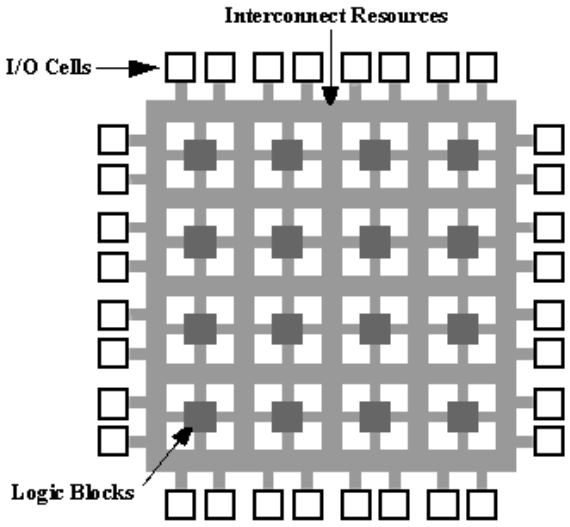


Figure 4.1 FPGA Internal Architecture [25].

In particular, each logical block contains some inputs of a Look Up Table (LUT) and a Flip-Flop. These inputs are arranged on the four sides of the block, and the various blocks may be linked together through a switch where the routing table can be programmed by user. The main benefits resulting from the use of these devices are linked to the cost of manufacturing. In fact, such devices can be mass produced at low cost, because of the manufacturer should not be worried to implement a particular functionality on them, as was the case for Application Specific Integrated Circuit (ASIC) devices. Compared to the latter, FPGAs are programmable directly from the end-user, who may decide to modify or correct errors simply by reprogramming the device at any time.

The differences between FPGA and ASIC technology are reported below:

FPGA	**ASIC**
Field Programmable Gate Array	**Application Specific Integrated Circuit**
• Faster design cycle.	• Longer design cycle.
• Correction of errors.	• Correct errors are not possible.
• Device costly.	• High initial costs, slow final chip cost.
• More area occupied.	• Long development time.

- Short development time.
- High power consumption.
- Low maximum operating frequencies

- Low power consumption.
- Less area occupied.
- Good maximum operating frequencies

A FPGA architecture, allows to give in a faster way the production phase with respect to an ASIC solution. Furthermore, using a reprogrammable approach is possible to intervene on the design after its completion, for example, in order to correct any errors and give improvements to it. An ASIC device has higher initial costs due to licensing and production of masks, but it has lower costs in the large-scale production of the same chip. However, these advantages in terms of the time and flexibility have also several disadvantages that reward ASIC solutions: an ASIC project presents better performance, in terms of power dissipation and occupied area because it is designed for a specific application. A FPGA is not optimized for a particular application, hence the efficiency of occupied area depends on the size of the project to implement. This last difference also limits the maximum operating frequency of the device, because it is in direct relation with the area occupied on it.

Among different FPGAs there are some distinction based on the technology, between EEPROM devices and S-RAM devices. The former are characterized by non-volatile memories capable of storing the logical file containing the map of the interconnections used. These FPGAs are generally small in size and are suitable for small-scale projects. Such technology provides programmable switch consisting of a transistor which can be enabled by injecting charge on the gate, in order to achieve a connection between the lines.

The FPGA that use S-RAM technology are based on pass gates: their behave is like switches. When the cell stores a low value, the transistor has a high resistance between the lines, preventing the connection. However, due to the volatile nature of the SRAM, the FPGAs must be configured at time of the implementation on the chip. Such devices therefore require a permanent external memory (EEPROM) for storing the mapping of interconnections. These FPGAs need more areas, resulting in a loss of performance in

terms of maximum operating frequency. However, they have a larger number of gates and I/O pins and therefore are suitable for the implementation of more complex circuits.

The main producers in the market of FPGAs are Xilinx and Altera, each able to offer different product families with completely different characteristics from the point of view of performance and costs.

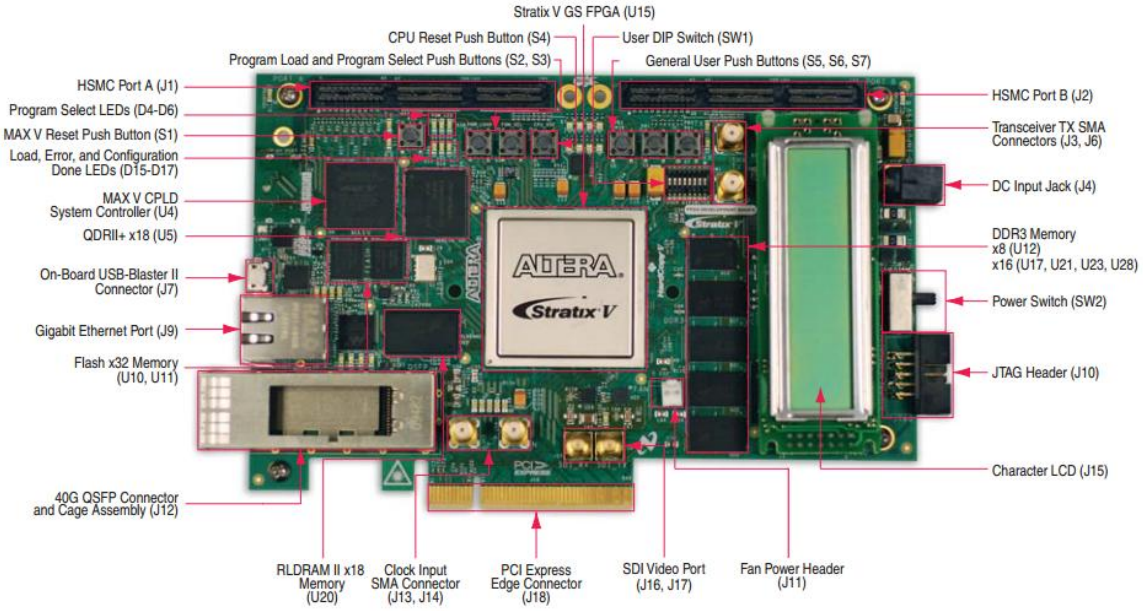


Figure 4.2: Altera Stratix V GS FPGA[26].

Over the years, the FPGA technology has been developed, becoming one of the best solutions in the field of programmable electronic technologies. They have reached performance and density levels increasingly, as the technologies to 65 nm (Stratix III), 40-nm (Stratix IV), until reaching today in technologies of 28 nm (Stratix V).

This work uses a Stratix V GS FPGA Altera family. This type of FPGA is optimized for high-performance, transceiver-based DSP-centric applications and high-bandwidth application designs. Among the main characteristics of interest to us, we have the development board includes a 50/100/125 MHz programmable oscillators, 864 user I/O, 1152-Mbyte of SDRAM with a 72-bit data bus and 172,600 Adaptive Logic Modules (ALMs)[26].

The general architecture of Altera, is based on a programming technology in SRAM, and is formed by an array of blocks (Programmable Logic Array Block) interconnected by a network of connections. These devices can be programmed using a Hardware Description Language (HDL) such as Verilog HDL (VHDL).

An electronic project can be composed of a large number of components, many of which are used in different parts of it, such as registers and logic gates. In order to facilitate the circuit design also complicated, the VHDL language gives the possibility of structuring our project in most of abstraction levels. The VHDL is a language for describing hardware through descriptions of the elements that make up the circuit capable of processing required.

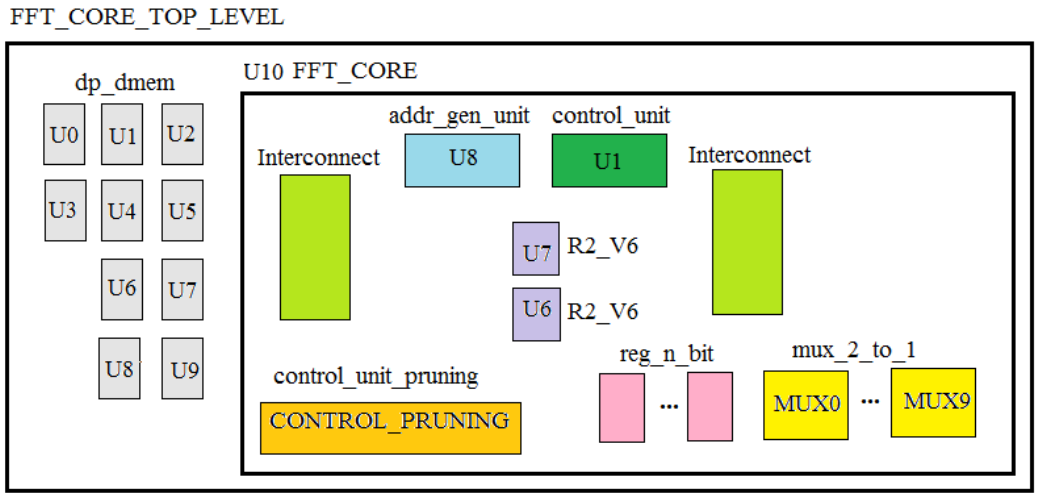


Figure 4.3: Structure of the project in VHDL. Block diagram of the components.

In order to identify the required components and to make easier the understanding of them, it is advisable to define macros embedded structures in which are inserted the individual elements, in order to work at different levels of abstraction. These structures are defined in general as entities, and they can characterize the operation of a single component or a block of many components. Figure 4.3 shows the block scheme used to implement our architecture FFT pruning. These macro structures are represented by FFT_CORE and FFT_CORE_TOP_LEVEL.

In hierarchical design it is possible to define the details of only one of the parts of the project at a time, test each part individually (allowing time debugging faster) and build the project in steps, inserting the various components one at a time.

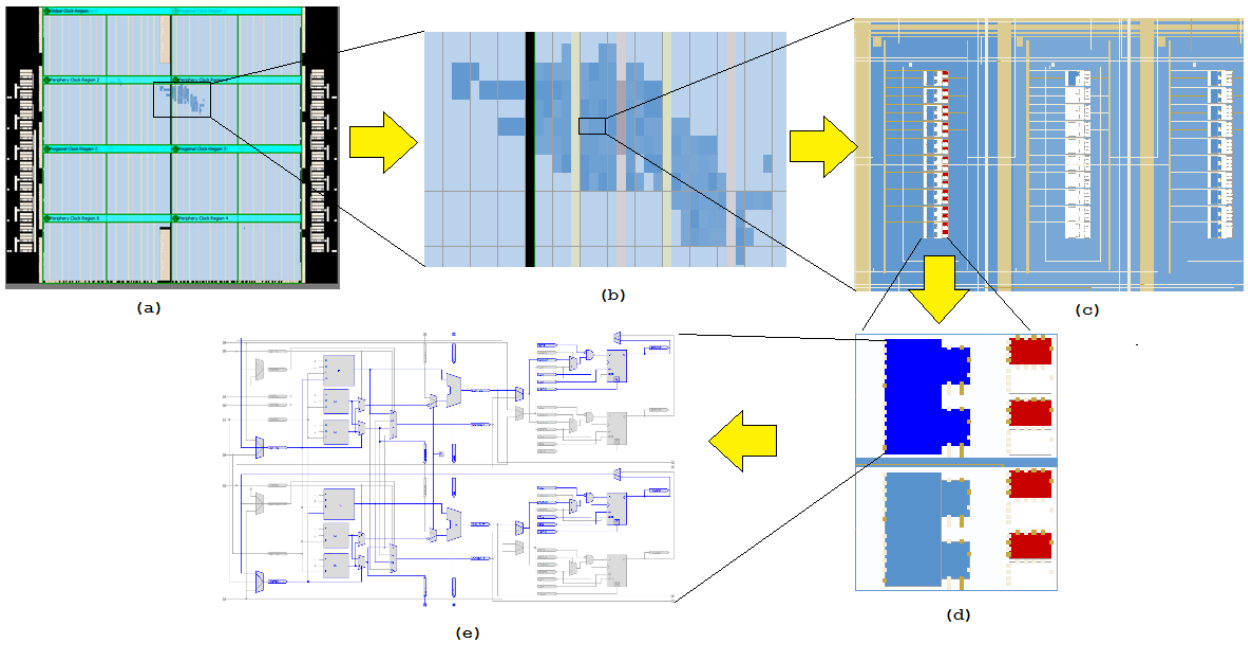


Figure 4.4: Chip Planner viewer: (a) seen from above of a FPGA after synthesis. (b,c,d,e) Zoom of the single logic elements of design.

In Figure 4.4 (a) the view of the design is reported. The internal structure of the FPGA is composed of many LC (b) interconnected with each other through communication routes can be observed. Inside of each LC a logical computations (e) are predefined but only a parts of its is used.

From RTL reported in Figure 4.5 view, the implementation of each single blocks inside the FFT architecture and all of the connections between them are shown.

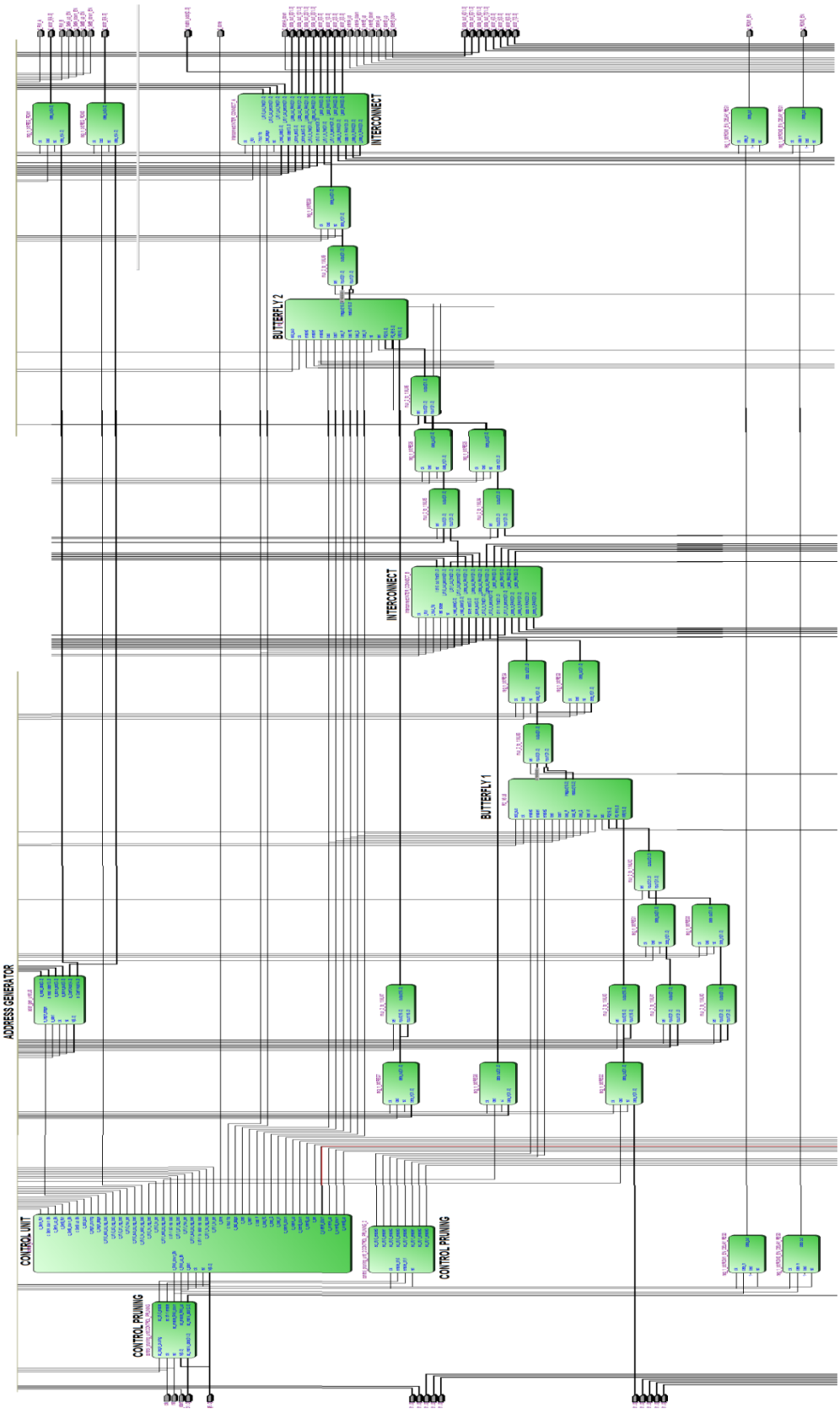


Figure 4.5: RTL viewer of entire architecture developed

4.2 Memory Pruning

The block of Pruning Matrix Memory is the first block that has to be defined in order to explain in a better way the structure of the Control Unit Pruning. This block contains the pruning matrix of our algorithm. One of the first questions that we made at the beginning was how to store this kind of matrix?

This memory is composed by *m* locations each of 32 bits. The number of columns of the pruning matrix is $n = log_2 N$ (like to the numbers of stages of Radix-2 algorithm) and for each column there are N/2 bits (one for each butterfly that we have to calculate).

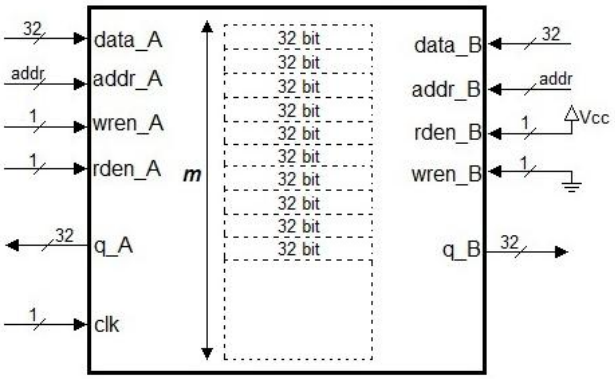


Figure 4.5: View of Memory Pruning component ports.

In order to store this matrix in a contiguous memory location of fixed length, we decided to adopt the following scheme: each column is mapped in each memory location of a fixed length of 32 bits. In this mode, the number of location m (and then the length of address signal) depends on the N points FFT, while the number of bits of each location is fixed. Clearly, having chosen to adopt a fixed length of each location at 32 bits, for FFT with N<64 the size of the memory coincides with the number of stages of the algorithm (m = n) . Instead, for N > 64, we need to define $m = n\,\frac{N}{64}$ because each column of the pruning matrix will be stored in N/64 memory locations. Although a variable length, we decided to dedicate at this component a maximum space of 2^9 locations.

In order to define the memory inside the FPGA, we used a block already implemented within the FPGA, through the megafunction *ALTSYNCRAM* provision by alters. In the component definition *ALTSYNCRAM*, we had to set various parameters, characteristic of our memory. As a first step, a new entity called *dp_dmem* is created, that indicated the generic memory block. The memory has to be connected with the FFT_CORE structure defined in this treatise, but also to be accessible from outside, in order to store the Matrix of our pruning. Then, a Dual Port Memory is chosen. This memory has two sets of ports (port A and port B) used for two-ports input / output, allows to access to memory twice at the same time. The "A" side of the memory is arranged to communicate with a block NETWORK, defined in order to interface our project with the external environment, while the side "B" is connected with the block CONTROL_PRUNING_UNIT in order to communicate with the structure FFT_CORE.

Our component will therefore generally consist of the following pins:

- `data_A(data_B)` for data input to port A(B) of the memory;
- `addr_A(addr_B)` that represent the address input to port A(B) of the memory;
- `wren_A(B)` *and* `rden_A(B)`: *to* write/read enable input for `addr_A(B)` port;
- `q_A (q_B)` that represent data output from port A (B) of the memory.

Following is the VHDL code contained in the file dp_dmem.vhd, which describes our memory:

```
ENTITY dp_dmem IS
   GENERIC (
      FILE_NAME    : STRING   := "file.txt";
      ADDR_WIDTH   : POSITIVE := 9;
      DATA_WIDTH   : INTEGER  := 32);


PORT(
      clk    : IN  STD_LOGIC;
      data_A : IN  STD_LOGIC_VECTOR (DATA_WIDTH-1 DOWNTO 0);
      addr_A : IN  STD_LOGIC_VECTOR (ADDR_WIDTH-1 DOWNTO 0);
      wren_A : IN  STD_LOGIC;
      rden_A : IN  STD_LOGIC;
      q_A    : OUT STD_LOGIC_VECTOR (DATA_WIDTH-1 DOWNTO 0);
      data_B : IN  STD_LOGIC_VECTOR (DATA_WIDTH-1 DOWNTO 0);
      addr_B : IN  STD_LOGIC_VECTOR (ADDR_WIDTH-1 DOWNTO 0);
      wren_B : IN  STD_LOGIC;
      rden_B : IN  STD_LOGIC;
      q_B    : OUT STD_LOGIC_VECTOR (DATA_WIDTH-1 DOWNTO 0));

END dp_dmem;
```

```
PRUNING_MATRIX_MEMORY: dp_dmem

    generic map(
      FILE_NAME    => "pruning_data_mem.txt",
      ADDR_WIDTH   => PRUNING_MEM_ADDR_WIDTH,
      DATA_WIDTH   => DATA_WIDTH)

    port map(
      clk     => clk,
      data_A  => data_in,
      addr_A  => addr_9,
      wren_A  => wren_9,
      rden_A  => wren_9,
      q_A     => data_out_9,

      data_B  => s_zero_data,
      addr_B  => s_matrix_addr,
      wren_B  => '0',
      rden_B  => '1',
      q_B     => s_matrix_data );
end rtl;
```

After the definition of the ports of our memory, the characteristics of this are defined:

- `OPERATION_MODE`: defines the type of memory that you want to use. This choice will depend on all the values of the other variables;
- `Width_a`: defines the number of bits constituting each memory location;
- `Widthhad_a`: defines the number of bits of the address bus;
- `Numwords_a`: defines the number of words stored in memory ($= 2^{addr\_width}$);
- `Outdata_reg_a`: defines the clock used for the synchronization of the output ports A; there are also `rdcontrol_reg, address_reg, indata_reg` etc.…
- `Byteena_a`: Byte enable input to mask the `data_a` port so that only specific bytes, or bits of the data are written;

- `Intended_device_family` and `lpm_type`: set the type of FPGA device we want to use;

```
altsyncram_component : altsyncram

      GENERIC MAP (
         operation_mode                          =>"BIDIR_DUAL_PORT",
         width_a                                 => 32,
         widthad_a                               => addr_width,
         numwords_a                              => num_words,
         outdata_reg_a                           => "CLOCK0",
         outdata_aclr_b                          => "NONE",
         width_byteena_a                         => 1,
         width_byteena_b                         => 1,
         width_b                                 => 32,
         widthad_b                               => addr_width,
         numwords_b                              => num_words,
         rdcontrol_reg_b                         => "CLOCK0",
         address_reg_b                           => "CLOCK0",
         indata_reg_b                            => "CLOCK0",
         wrcontrol_wraddress_reg_b               => "CLOCK0",
         outdata_reg_b                           => "CLOCK0",
         read_during_write_mode_mixed_ports      => "DONT_CARE",
         ram_block_type                          => "AUTO",
         init_file                               => FILE_NAME,
         clock_enable_input_a                    => "BYPASS",
         clock_enable_input_b                    => "BYPASS",
         clock_enable_output_a                   => "BYPASS",
         clock_enable_output_b                   => "BYPASS",
         intended_device_family                  => "Stratix V",
         lpm_type                                => "altsyncram",
         power_up_uninitialized                  => "FALSE"

         )
```

Those just presented are some of the most important parameters that we defined in the component *altsyncram*.

Some words about simulation test bench

In order to simulate the results of our algorithm in Modelsim, the contents of the memory of pruning has to be defined through a definition of text files included inside the project

folder. In order to read the text file we have defined the following function *initialize_ROM* that returns an array called *ROM_memory:*

```
architecture rtl of dp_dmem is

  subtype rom_word      is std_logic_vector(DATA_WIDTH-1 downto 0);
  type    ROM_memory    is array ((2**ADDR_WIDTH)-1 downto 0) of rom_word;
  file    file_handle   : text;


impure function initialize_ROM(file_name : string)

  return ROM_memory is

    variable ROM_mem_init     : ROM_memory;
    file     file_handle      : text;
    variable lineread         : line;
    variable dataread         : integer;
    variable is_open          : boolean:= false;
    variable index            : integer := 0;

    begin

    if is_open = false then

        file_open(file_handle, file_name, READ_MODE);
        is_open := true;

    end if;

    index := (2**ADDR_WIDTH);

    for i in 0 to (conv_integer(index)-1) loop

        if (not endfile(file_handle)) then

            readline(file_handle, lineread);
            read(lineread, dataread);
            ROM_mem_init(i) := conv_std_logic_vector(dataread, DATA_WIDTH);

        end if;

    end loop;

    file_close(file_handle);
    return ROM_mem_init;

  end function initialize_ROM;
```

Thanks to this function which refers to the system function in the package *std.textio.all*, a text file is read, and associate its content to the content of a memory; a number of memory locations corresponding to the number of lines in the text file. ($= 2^{\text{ADDR\_WIDTH}}$). This function is called at each clock cycle through the follow reading process called READ_PROCESS:

```
signal ROM_memory_bank : ROM_memory  := initialize_ROM(FILE_NAME);
signal undef_value     : std_logic_vector(ADDR_WIDTH-1 downto 0) := (others => 'U');
begin

  READ_PROCESS: process(clk)
  begin

      if clk'event and clk = '1' then          Read only if the address is valid!
          if rom_addr /= undef_value then
              rom_data_out <= ROM_memory_bank(conv_integer(unsigned(rom_addr)));
          else
              rom_data_out <= (others => 'Z');
          end if;
      end if;                                  Read from memoery and send the data in
                                               the output bus of the memory!
  end process READ_PROCESS;
```

4.3 Control Unit Pruning

After the definition of the method to implement the *Pruning Memory* in VHDL, the declaration of Pruning Control Unit block is presented. This control unit is divided into two distinct blocks called *Control_Pruning_Unit* and *Control_Pruning_Unit_2*, each of which has precise tasks. Figure 4.6 shows the definition of two blocks entity, with all input and output ports declared for each of these.

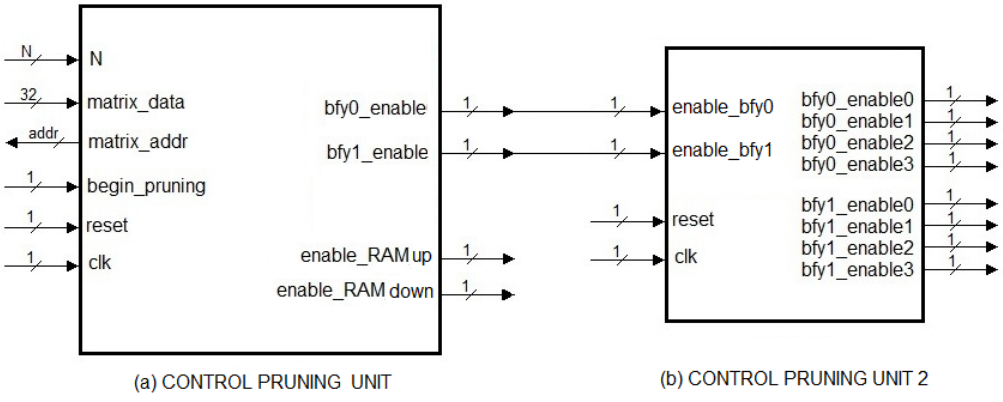


Figure 4.6: Entity blocks Pruning Control Unit and Control_Pruning_Unit_2.

```
entity control_pruning_unit is

  generic (
    N_width              : integer := 5;
    DATA_WIDTH           : integer := 32;
    ADDR_WIDTH           : integer := 3);

  port (
    clk                  : in  std_logic;
    rst                  : in  std_logic;
    N                    : in  std_logic_vector(N_width-1 downto 0);
    ac_begin_pruning     : in  std_logic;
    ac_matrix_data       : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    ac_matrix_addr       : out std_logic_vector(ADDR_WIDTH-1 downto 0);
    ac_enable_RAM_up     : out std_logic;
    ac_enable_RAM_down   : out std_logic;
    ac_bfy0_enable       : out std_logic;
    ac_bfy1_enable       : out std_logic);

end control_pruning_unit;
```

(a)

```
entity control_pruning_unit_2 is

  port (
    clk              : in  std_logic;
    rst              : in  std_logic;
    enable_bfy0      : in std_logic;
    enable_bfy1      : in std_logic;

    bfy0_enable0     : out std_logic;
    bfy0_enable1     : out std_logic;
    bfy0_enable2     : out std_logic;
    bfy0_enable3     : out std_logic;
    bfy1_enable0     : out std_logic;
    bfy1_enable1     : out std_logic;
    bfy1_enable2     : out std_logic;
    bfy1_enable3     : out std_logic);

end control_pruning_unit_2;
```

(b)

Figure 4.7: VHDL code of Pruning Control Unit (a) and Control_Pruning_Unit_2 (b) blocks.

The *Control_Pruning_Unit* block (Figure 4.7(a)) has the task of interfacing with the pruning memory and to produce four output enable signals. The *Control_Pruning_Unit_2* block instead, has the task of producing only the internal enable signals for each butterfly.

4.3.1 Pruning Algorithm Implementation

In order to understand better how the logic of control is defined, Figure 4.8 is shown. The algorithm of pruning can be divided in four main steps:

1. ***Reading of pruning memory phase***;
2. ***The data processing phase;***
3. ***Delay phase;***
4. ***Dubbing phase***;

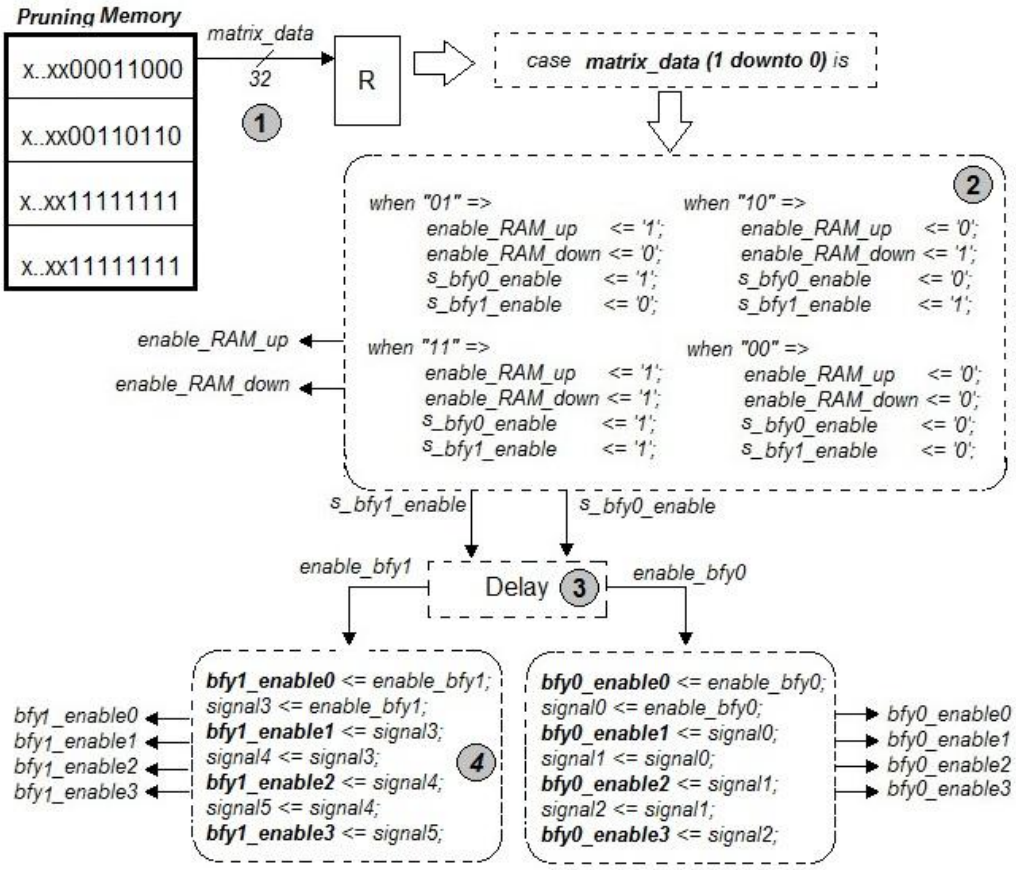


Figure 4.8: Block diagram of the four phases of the Control Pruning Unit logic.

1. *Reading of pruning memory Phase*:

The step of reading from the pruning memory starts when the signal *start_pruning* comes from the FFT core. At this moment, the control unit generates the memory address signal called *matrix_addr*. It is a binary signal composed by nine bits defined by the PRUNING_MEM_ADDR_WIDTH constant; during the design, a RAM memory with 2^9 locations of 32 bits each is adopted: in this way, FFT up to 2048 points can be implemented. The address signal is sent to the pruning memory to select the memory location from which takes the data. In order to generate this address are used a simple incremental counters, which are defined by setting two variables: ROW_FINISH and NUM_ROWS_PER STAGE.

The defined memory has locations with fixed length of 32 bits (4 bytes) that contain the columns of the pruning matrix. The ROW_FINISH constant indicates the number of butterflies present in each stage, i.e. the number of bits of each column of the pruning matrix. Therefore, in order to better understand how to set the constant *ROW_FINISH* some consideration are introduced. In case of FFT with N > 32, the number of butterflies present in each stage are greater than 32, so, for mapping each column of the pruning matrix, more memory locations are needed. For this reason, *ROW_FINISH* = 32 (it is the maximum). For N = 16,32 the constant *ROW_FINISH* is set as N/2 because only the first N/2 bits (of the 32 bits that composed each location) will carry information.

Defined the number of bits that carry information for each memory location, the number of memory locations that we have to use to store each column of the pruning matrix have to be defined: for N = 16 or 32 or 64, the constant *NUM_ROW_PER_STAGE* is set as 1, while for N > 64, the *NUM_ROW_PER_STAGE* is N/64. Since, for the considerations made previously, more rows of the memory pruning to map N / 2 butterfly of each stage are needed. Defined the structure of memory, a counter generates the signal *matrix_addr* simply by increasing its value in each new reading. In particular, if N = 16 or 32 or 64 the counter is incremented only at the beginning of each stage (indicated by the signal *begin_stage*). In the case of N > 64 it is necessary to increase *matrix_addr* more times inside each stage, since the mapping of the butterflies in each stage will be contained in most memory locations of pruning (NUM_ROW_PER_STAGE). In this case, the problem of access to the pruning memory and load the data in a continuous mode has to be solved. Introducing a register is possible to store the contents of a memory location and, in the same time, to access at the next memory location.

The Figure 4.9 shows an example of a pruning memory for FFT with N = 128; the figure reports also the reading phase of the first line and the initial part of the second phase, that will be presented in the next point.

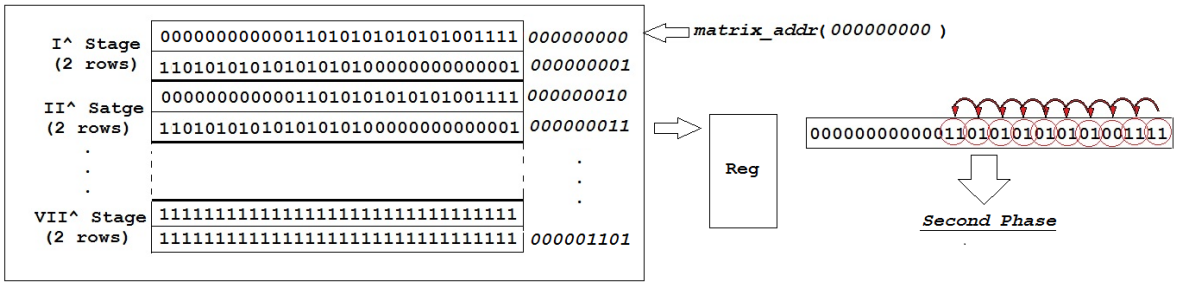


Figure 4.9: Example of reading phase for N=128.

2. ***The data processing phase***

The second phase shown in Figure 4.8 is the heart of the computation of the pruning. In this phase, the data from the memory is processed and four output signals are generated: *enable_RAM_up*, *enable_RAM_down*, *s_bfy0_enable*, *s_bfy1_enable*. Each bit of the signal *matrix_data* represents the status of the two input samples of a butterfly: if both input signals to a butterfly are zeros then the corresponding bit of the *matrix_data* will be zero. Conversely, if at least one of the input samples to the butterfly is different from zero, the bit corresponding is one. As already discussed previously, the value of this bit talks if a butterfly must be considered in the calculation of the FFT.

Since the FFT architecture is composed of two butterflies blocks able to work simultaneously, the phase of processing is carried through a construct "case" taking two bits at a time signal *matrix_data*. For example, if *matrix_data = "01"*, means that the second butterfly has zero inputs, therefore it should be disabled:

```
s_bfy0_enable   <= '1';
s_bfy1_enable   <= '0';
```

Furthermore, it is possible to identify the RAM memory banks where the input samples of each butterfly are stored. The first butterfly always refers to the memory banks *Ram_0* and *Ram_1* or *Ram_4* and *Ram_5* depending on the processed stage. The second butterfly takes its inputs from the memory banks *Ram_2* and *Ram_3* or from the banks *Ram_6* and

Ram_7 depending on the stage that is calculating. Therefore, from the knowledge that a certain butterfly has its inputs both zero it is possible to disable the reading from the corresponding memory bank, and save the energy needed for its access, via signals:

```
enable_RAM_up     <= '1';
enable_RAM_down   <= '0';
```

The processing of the bits of the signal *matrix_data* has to been done continuously and synchronously with the generation of addresses by the *Address Generation Unit* of the FFT algorithm. Every two cycles of clock, for the entire time period of reading, a flag counter increments of two values, in order to process two bits on the signal *matrix_data* at a time.

The signals produced by this phase have two different destinations: enable signals of RAM memories are sent immediately to the memories through the *Control Unit* of the FFT; instead, signals *s_bfy0_enable* and *s_bfy1_enable* are sent to a further step.

3. Delay Phase

The *Delay Phase* has the task of generating two signals (*enable_bfy0* and *enable_bfy1)* which are a delay version of the two signals *s_bfy0_enable* and *s_bfy1_enable*. This step is necessary because the pruning algorithm must be able to respect the deadline of algorithm FFT.

The samples from the memory banks take four clock cycles to arrive at the entry of each *butterfly block.* Hence, the enable signal of a butterfly are produced at the same clock of arrival of the samples. Figure 4.10 shows the chain of four latches used to implement this Delay block.

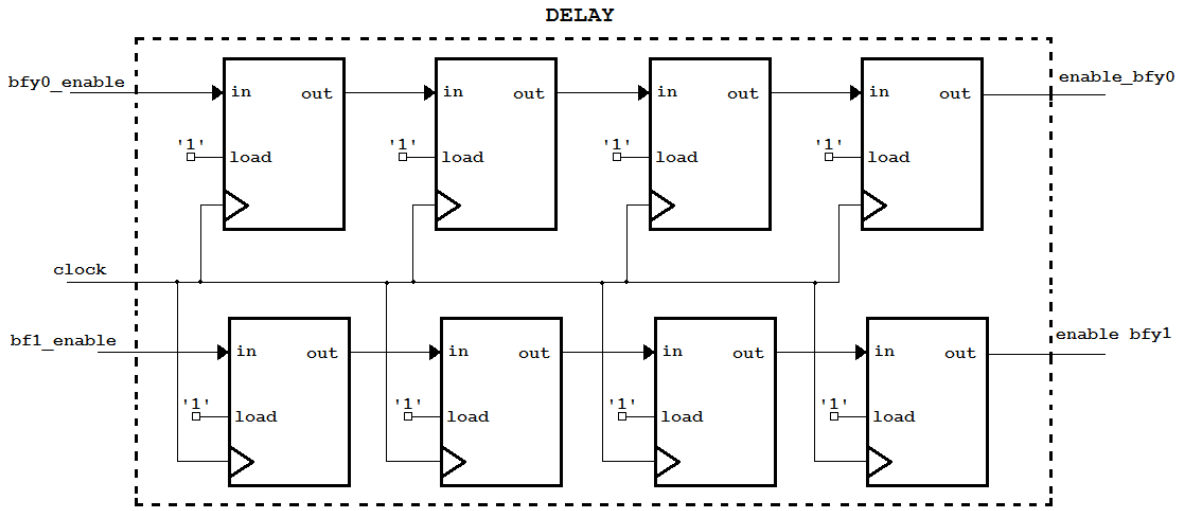


Figure 4.10: Delay network of latch

4. Dubbing Phase:

The signals *enable_bfy0* and *enable_bfy1* generated by the delay network are directed to the two butterflies. As presented in Figure 3.3, each butterfly is pipelined, allowing the executions of the arithmetic operations in a sequential mode. Each step of the butterfly's pipeline requires its enable signal. In order to produce these enable signals, an additional entity called *Control_Pruning_Unit_2* is introduced. This entity takes the two *enable_bfy0* and *enable_bfy1* signals from the *Control_Pruning_Unit* and it generates four enable signals required for each butterfly. In the Figure 4.11 on the left the VHDL code implemented inside the entity *Control_Pruning_Unit_ 2* is shown, while on the right we find the corresponding waveform of enable signals produced.

In Figure 4.11.1, the RTL view of the pruning part that implements the dubbing phase is reported.

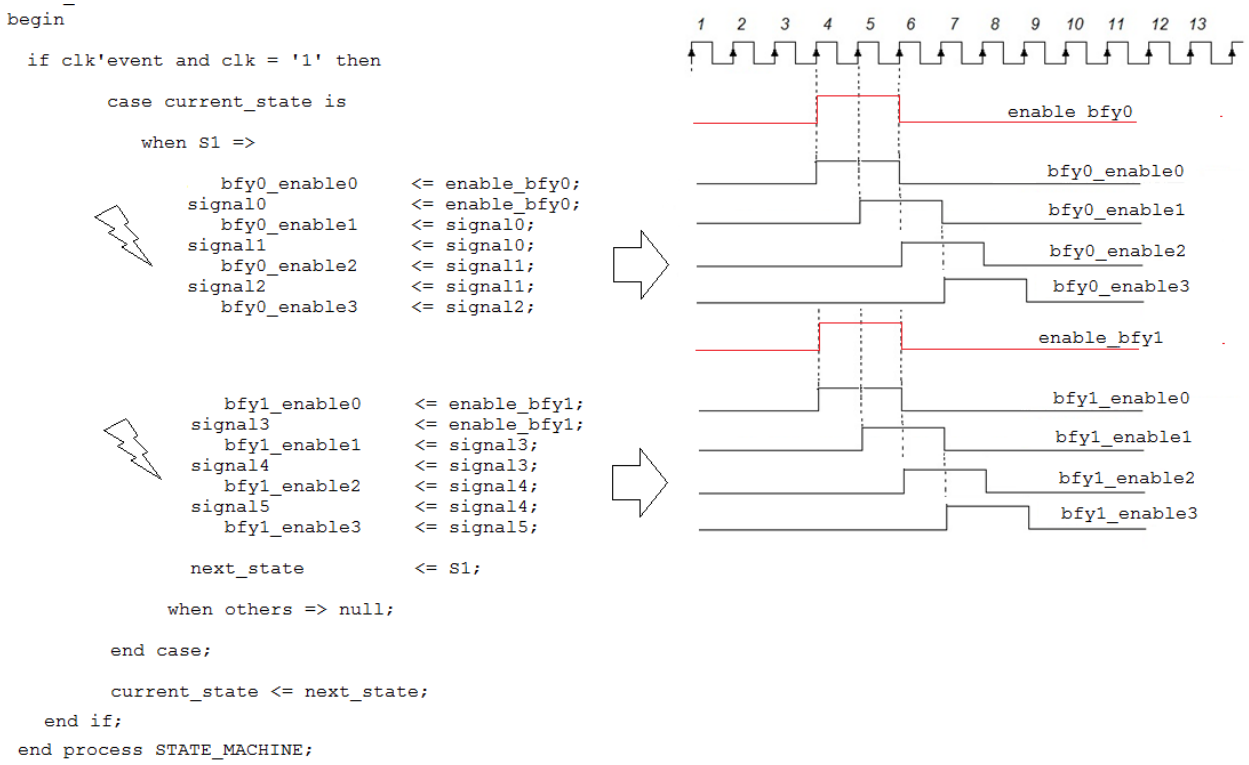


Figure 4.11: Algorithm implemented in Control_Unit_Pruning_2.

4.4 Network Interface

The last block is the *Network Interface*. This block was developed to facilitate the use of our architecture as IP in larger systems through intuitive interface.

The component's view is shown in Figure 4.12. This is inserted between the user and our FFT block pruning. The term of "user" indicates all those systems which are adapted to acquisition of data from the channel and preceding the FFT block in OFDM receiver system. On the basis of the FFT points, the user can set three parameters:

- *WIDTH_N* is dependent on the N number of inputs to the FFT and it is always calculated as $(\log_2 N) + 1$.

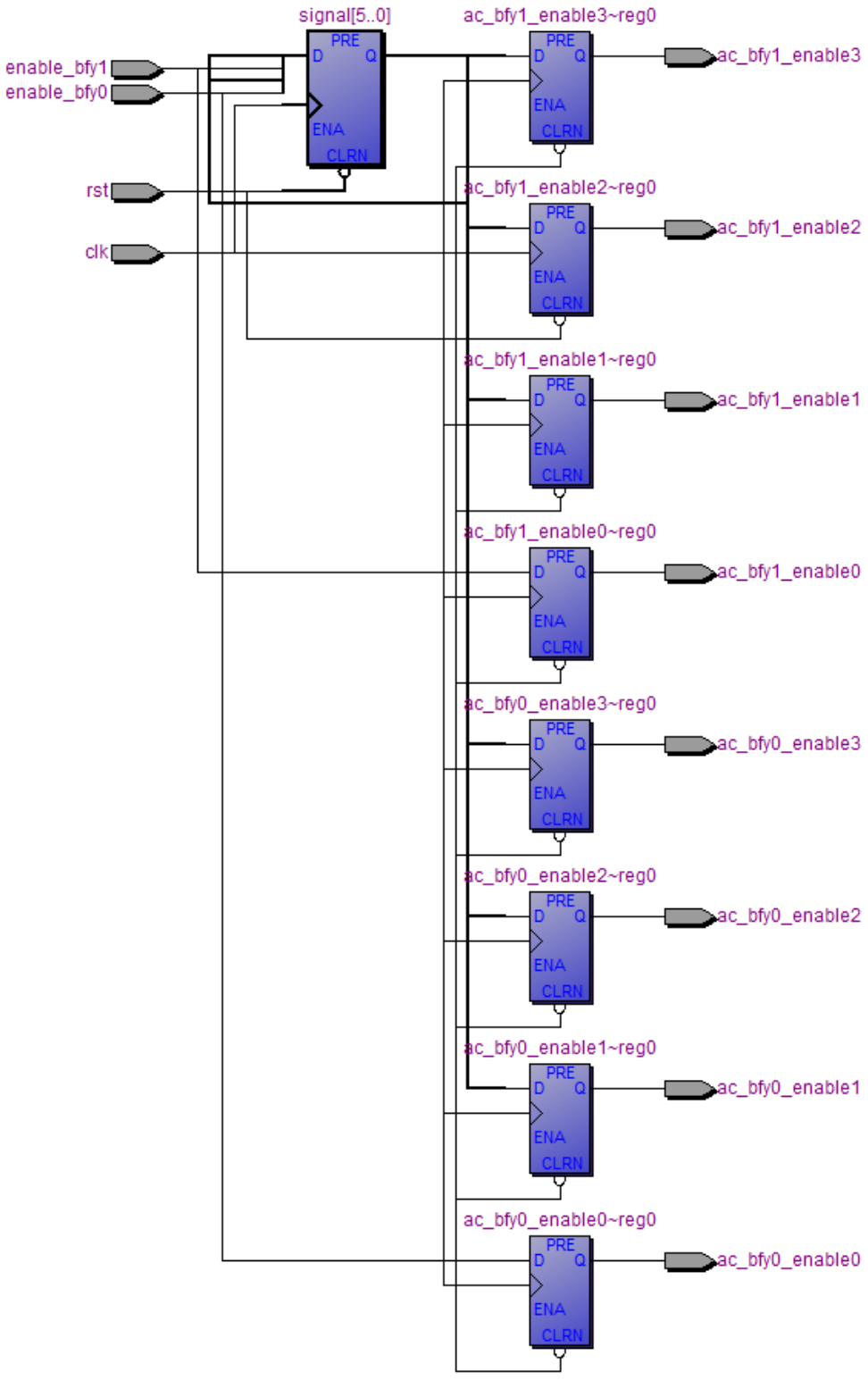


Figure 4.11.1: RTL View of the final part of control pruning unit (Dubbing Phase)

- *ADDR_WIDTH* it also depends on the number of points N, and represents the number of bits needed for the addressing of all memory banks present in the proposed architecture.

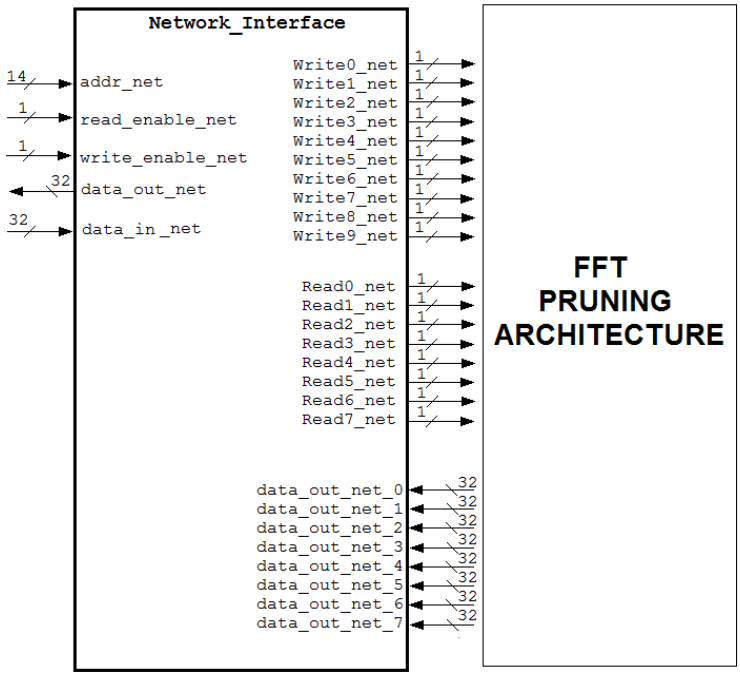


Figure 4.12: Network Interface block.

For example, considering an architecture with $N = 2048$, four bits (2^4 $combinations$) for addressing the ten memory banks (eight banks of RAM, one bank of memory for Twiddle Factor and one pruning memory) and ten bits for addressing the individual locations located inside of each of them, are required. The memory banks have the same address signal, and the maximum length of them is decide by Twiddle Factor memory (N/2 locations = 1024 = 2^{10}).

- DTA_WIDTH = 32 (bits) to set the length of each sample.

The user can send the input samples and decide to store them in the memory bank with the proper addressing *addr_net*. Moreover, with the two *write_enable* and *read_enable* signals it is possible to communicate the intent of read/write from/to selected memory banks.

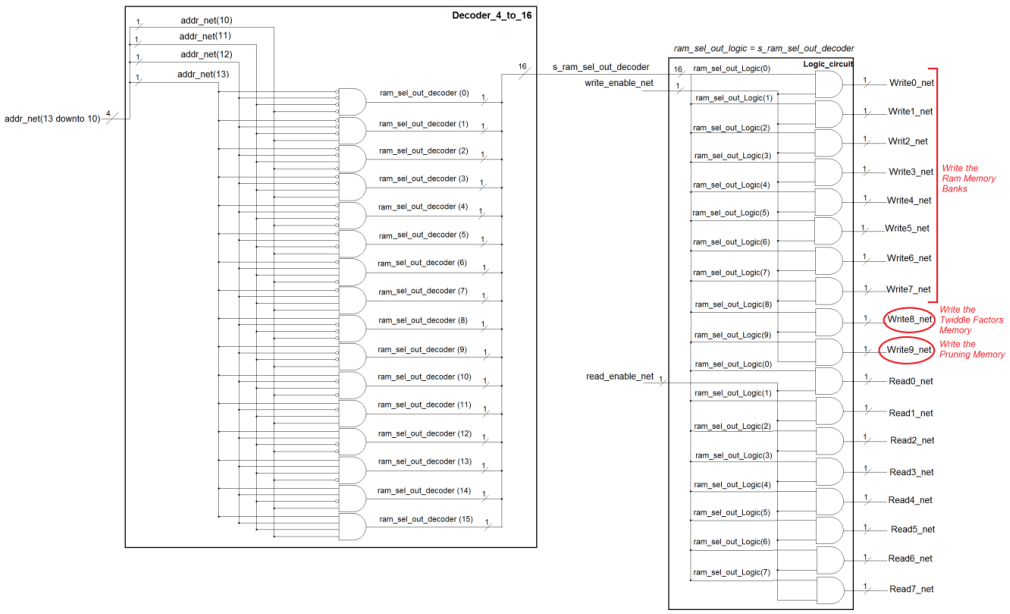


Figure 4.13: Configuration of blocks inside the Network Interface block.

In the Figure 4.13 are represented the blocks that we have defined inside the *Network Interface* block. At the output of these, 10 signals *Write*i*_net* and 8 signals *Read*i*_net* each of which is addressed to each memory bank are presented. (*Ram_0, Ram_1, Ram_2, Ram_3, Ram_4, Ram_5, Ram_6, Ram_7, Twiddle_Factor_Ram, Ram_pruning*) Furthermore, via the signals *data_out_net_*i** the user can read the samples from each ram memory bank. The memory of pruning is a type of memory that no need to be read by external user, then it doesn't require the read signal.

The first block of the Figure 4.13 is a *decoder_4_to_16:* it takes the last four bits of *addr_net* and produce a signal of enable ram. The first's ten bits of the address signal *addr_net*, are sent directly to the memories of architecture in order to addressing each location inside of them. The next block called *Logic_circuit* takes this signal and through the signals write and read, it enables the memory banks.

CHAPTER 5

Results

The pruning architecture is proposed to decrease the average power dissipated by the FFT algorithm. In order to evaluate the results of power consumption we used *Quartus II* environment (Version 12.1). The simulation was performed on an *FPGA Stratix V GS 5SGSMD5K2F40C2N* device with the characteristics shown in Table 1. As operating frequency we selected 100 MHz. For N = 2048 the computation time for the FFT is equal to 120 μs in respect of the IEEE standard using for DAB, DVB-T wireless transmissions. With the same order of FFT but, with a frequency of 200 MHz the FFT period drops to 56 μs in respect also of 3GPP-LTE standard.

Device-Family	**Stratix V GS 5SGSMD5K2F40C2N**
Vcc	0.90 [V]
Frequency	100 [MHz]
Time of Simulation	120 [us]
Total Pins	680/864
DSP blocks	4
Logic Utilization (in ALMs)	1056/172600
Total LC Combinational	1170
LC Combinational by FFT core	1057
LC Combinational by pruning	113

Table1: Characteristics of FPGA used.

Table 1 also shows the number of *Logic Cell* (LC) used for implement the logic of pruning. As showed in Figure 5.1, the introduction of two additional blocks for the execution of pruning results in an increase of 10% approximately of the number of LC required with respect to

sample FFT architecture. This increase is explained by the presence of the additional block *Pruning Control Unit* and its internal logic.

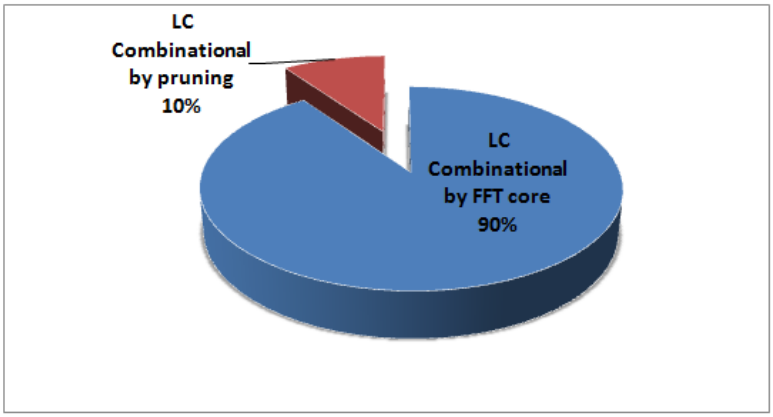


Figure 5.1: LC combinational used.

The reduction of power consumption was evaluated using the *Power Play Altera Analyzer* tool, to compare different profiles of pruning to a *Radix-2 FFT* non-pruned. These profiles are shown in Figure 5.2: each string is 2048 samples in the input, while each cell in dark gray represents 32 samples assumed equal to zero. The pattern *a* represents the base case of input vector without any zero samples, hence it is the reference case of power consumption. Among the several patterns used for the simulations, two categories can be identify: the first type consists of consecutive zeros distributions (*pattern b, c, d, e, f*). The second type examines zeros distributed uniformly along the input vector (*Pattern g, h, i, l, m*).

This reduction of power dissipation depends on the number of zeros in the input but also depends on the type of distribution. For example, the pattern *d* and *h* use the same number of zeros input but show two different power results. This is justified by the fact that the implemented pruning algorithm, refers to a particular order of butterflies for each stage, and also each butterflies is excluded from the calculation only if both its inputs are zero. For this reason, the different distribution of zeros input, changes the degree of depth that the pruning reaches inside the algorithm. The most significant results are distinguished by the type of patterns *b, c, d, e, f* of the figure, which provide distributions of consecutive zeros.

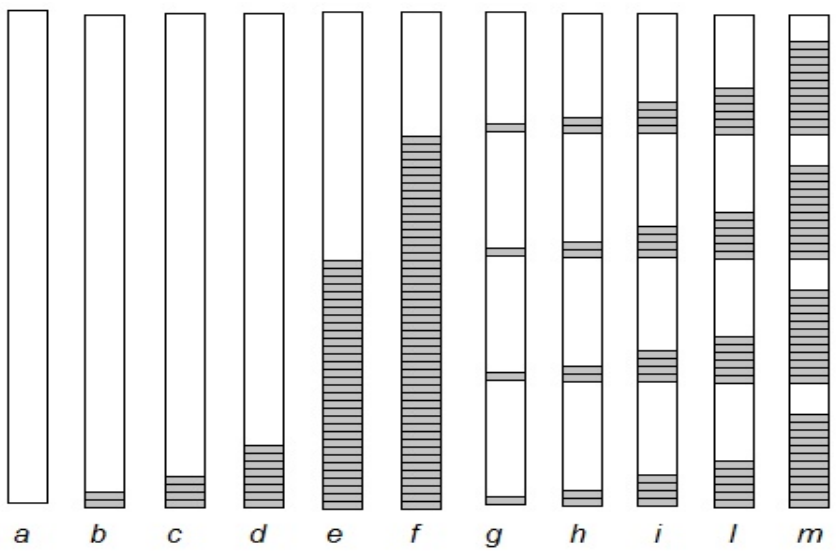


Figure 5.2 Pattern of zero inputs.

Table 1. Power Consumption with different pattern of Zero Input

Pattern Zero	Total Power by Hierarchy	Percentage of Power Saved
Pattern a	182 [mW]	–
Pattern b	146 [mW]	19.8 %
Pattern c	144 [mW]	20.9 %
Pattern d	137 [mW]	24.7 %
Pattern e	116 [mW]	36.3 %
Pattern f	106 [mW]	41.8 %
Pattern g	183 [mW]	-
Pattern h	177 [mW]	2.7 %
Pattern i	162 [mW]	11.0 %
Pattern l	138 [mW]	24.2 %
Pattern m	128 [mW]	29.7 %

Table 2: Results of power saved by different pattern of zero.

In particular, for the distribution *g* the dissipated power is even greater than in case *a* (reference case). The power is slightly increased. This atypical behavior theoretically, can be justified by the presence of the logic of pruning that consumes power to produce the enable signals in the first stage. With the pattern *g*, the energy saved by pruning is lower than the energy used to generate enable signals of pruning and also because the level of

accuracy obtained from the *Quartus II* analysis is not optimal in order to evaluate finely the power consumption.

However, the best results are obtained for long distributions of zeros, when the depth of pruning can reach good levels of the algorithm. In a real scenario this is the most frequent case. As described in a IEEE 802.22 standard [27], in a typical TV band scenario, the pattern of TV channels occupancy by users over time and over the frequency are distributed as shown in Figure 5.3:

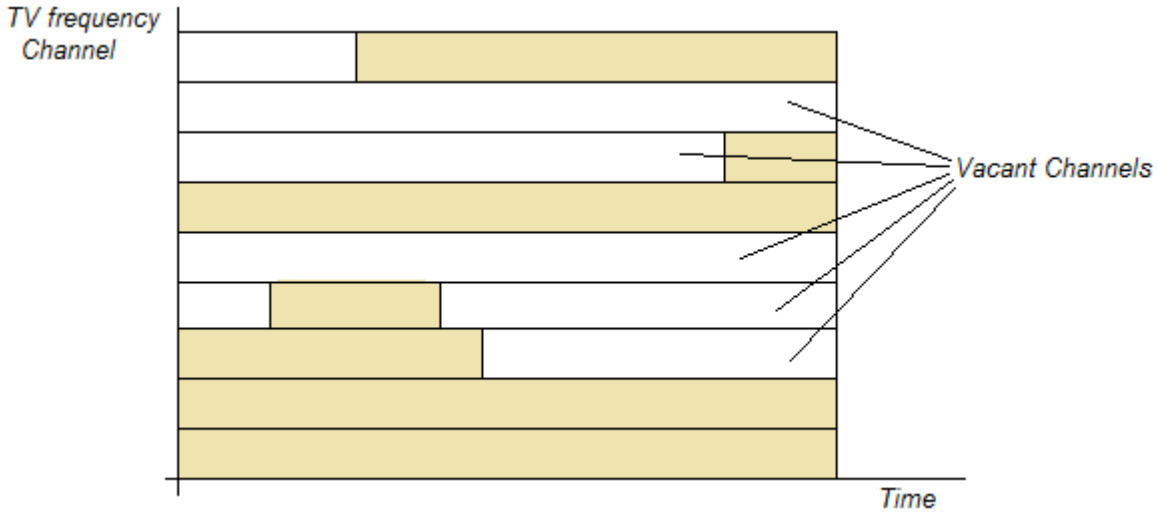
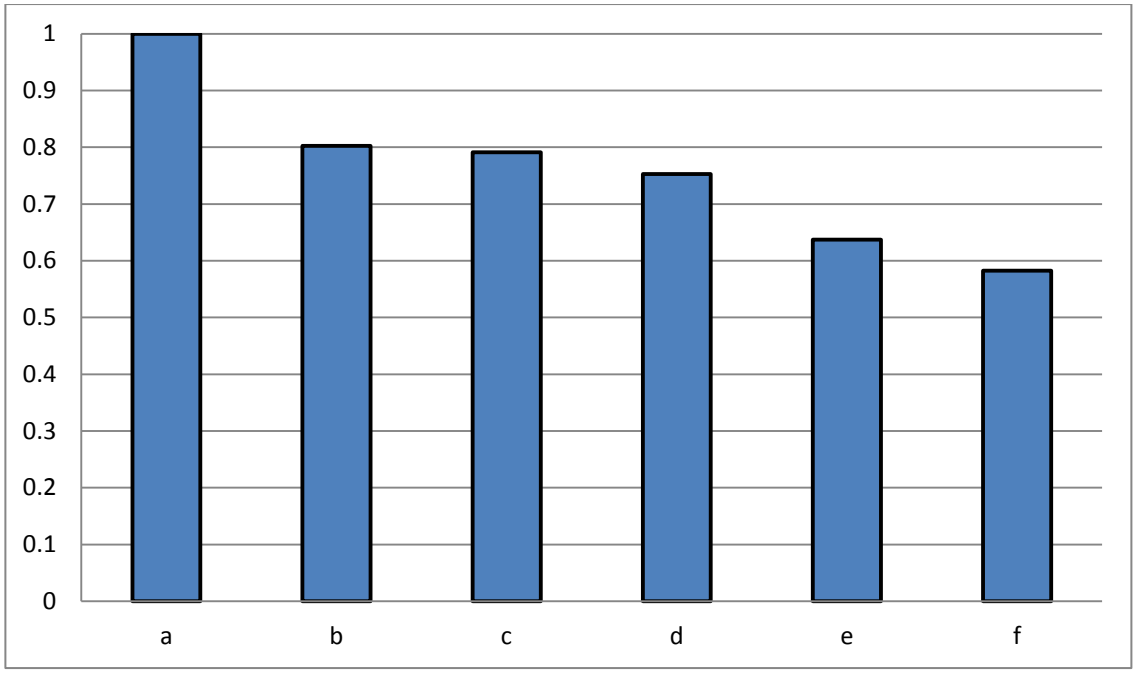


Figure 5.3: Example of TV band occupancy of different channels [27] .

These distributions represent that in this real TV scenario where a Radio Cognitive receiver can find application , the most frequency patterns topology have long sequence of contiguous zero samples.

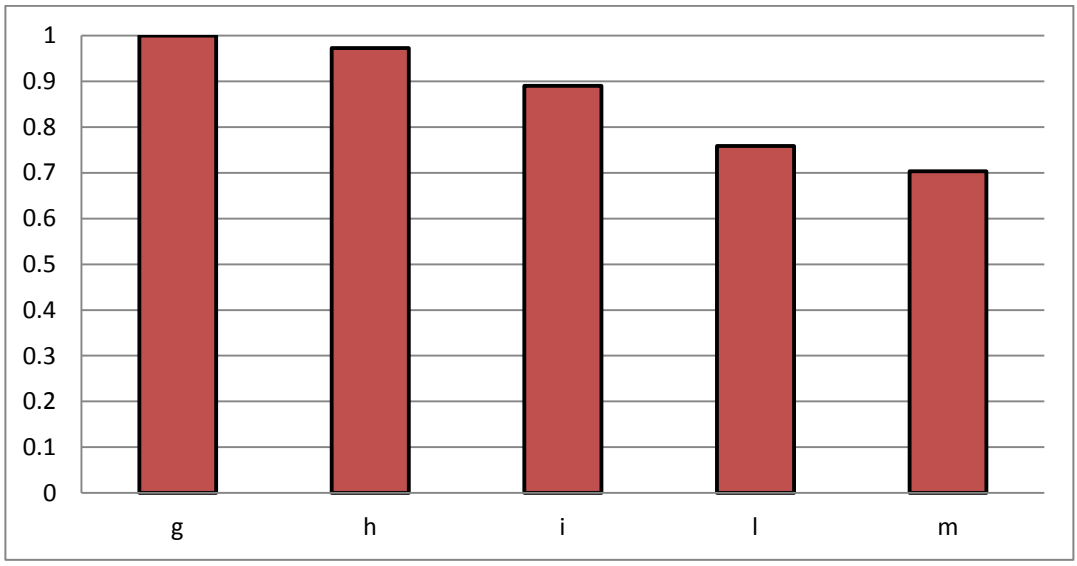
The graph 1 and 2 report the results of the power consumption by the architecture FFT, normalized with the non-pruning case (pattern *a*). For both graph, the patterns *a* and *g* are take as reference patterns, assuming them with the same power consumption normalized at 1.

.



Graph 1 : Total Power by Hierarchy normalized with the non pruning case

Contiguous pruning distribution



Graph 2: Total Power by Hierarchy normalized with the non pruning case

Uniform pruning distribution

Conclusion

At the input of a receiver block NC-OFDM for Cognitive Radio applications, there are often long sequences of zeros because the transmission from the users can be distributed non-uniformly. This research work presents the design and implementation of a FFT pruning block, which is an extension to the FFT cores for OFDM demodulation, enabling run-time pruning of the FFT algorithm, without any restrictions on the distribution pattern of the active/inactive subcarriers.

The pruning block is added to a FFT architecture based on an FFT Radix-2 algorithm, in order to deactivate the redundant operations in correspondence of zeros input. The design and implementation of FFT processor core is not the part of this work. The proposed pruning algorithm is based on the definition of a binary matrix, where inputs of the butterflies that compose each stage of the Radix-2 algorithm are mapped. Through a clock gating technique, the architecture can disable the arithmetic operations between zeros (as multiplications and additions), that are locate inside the butterflies blocks of FFT. The task of the *Control Pruning Unit* is to interpreting the matrix and producing the enable signals that are sent to the butterfly's blocks.

The implemented pruning algorithm is not proposed to improve the time occurred by FFT algorithm but only to decrease its computational work. However, the time required to perform the calculation of a 2048-FFT is in according with the specifications imposed by the IEEE standard for DAB, DVB-T and 3GPP-LTE wireless transmissions.

The whole design was prototyped on an Altera Stratix V FPGA to evaluate the performance of the engine pruning. Synthesis and simulation results showed that the logic overhead Introduced by the pruning block is limited to 10% of the total resources utilization. In order to evaluate the performance in terms of power dissipated, the tools Power Play Analyzer of Quartus II are used, for different patterns of zero input to a 2048-

FFT. The type of patterns represents the majority of the zero distributions obtainable in input to a NC-OFDM demodulator; in particular, continuous and distributed pattern of zero are chosen.

The results showed that in the presence of a medium-high scattering of the sub-carriers, power and energy consumption of the FFT core were reduced by 30% factor. In the best cases, it was possible to obtain a power reduction by 40% factor.

The results of power saving does not show a linear correlation with the number of zeros samples in input because it depends strongly on the type of the pattern chosen and on the depth with it propagates inside the Radix-2 algorithm.

Bibliography

[1] *Staple, G.; Werbach, K.; , "**The end of spectrum scarcity [spectrum allocation and utilization]**," Spectrum, IEEE , vol.41, no.3, pp. 48- 52, March 2004*

[2] ***NeXt generation/dynamic spectrum access/cognitive radio wireless networks**: A survey – Computer Networks Volume 50, Issue 13*

[3] *Noam, E.M.; , "**Taking the next step beyond spectrum auctions: open spectrum access ,**" Communications Magazine, IEEE , vol.33, no.12, pp.66-73,*

[4] *Federal Communications Commission, **"Spectrum Policy Task Force**" Rep. ET Docket no. 02-135, Nov. 2002.*

[5] *J. Mitola et al., "**Cognitive radio: Making software radios more personal**" IEEE Pers. Common., vol. 6, no. 4, pp. 13–18, Aug. 1999*

[6] *Federal Communications Commission (FCC). ET docket no. 03-322. Notice of Proposed Rule Making and Order, December 2003*

[7] *"Fischer, Walter. Digital television: a practical guide for engineers. Springer, 2004.*

[8] *Sembiring, Z.; Syahruddin, M.; , "**Performance Analysis of Discrete Hartley***

***Transform Based OFDM Modulator and Demodulator**," Intelligent Systems, Modelling and Simulation (ISMS), 2012 Third International Conference on , vol., no., pp.674-679, 8-10 Feb. 2012*

[9] 01CXGBN –Trasmissione numerica
http://www.tlc.polito.it/garello/trasmissione/L19.pdf

*[10] James W.Cooley and John W. Tukey "**An Algorithm for the Machine Calculation of Complex Fourier Series"** Vol. 19, No. 90, Apr., 1965*

*[11] Airoldi, R.; Anjum, O.; Garzia, F.; Nurmi, J.; Wyglinski, A.M.; , "**Energy-Efficient Fast Fourier Transforms for Cognitive Radio Systems**" Micro, IEEE , vol.30, no.6, pp.66-76, Nov.-Dec. 2010*

*[12] Parvez, Mohammad Zavid, Md Abdullah Al Baki, and Mohammad Hossain. "**Peak-to-Average Power Ratio Reduction in NC-OFDM based Cognitive Radio.**"International Journal of Signal Processing (SPIJ) 6.2 (2012): 5. –*

*[13] Duhamel, P.; Hollmann, H.; , "`**Split radix' FFT algorithm**," Electronics Letters , vol.20, no.1, pp.14-16, January 5 1984*

*[14] Douglas L. Jones. "**Decimation-in-time (DIT) Radix-2 FFT"** http://cnx.org/content/m12016/latest/*

*[15] Airoldi, R.; Garzia, F.; Nurmi, J.; , "**FFT Algorithms Evaluation on a Homogeneous Multi-processor System-on-Chip,**" Parallel Processing Workshops (ICPPW), 2010 39th International Conference on , vol., no., pp.58-64, 13-16 Sept. 2010*

*[16] Jarmo Takala and Konsta Punkka, Tampere University of Technology, P.O.Box 553, FIN-33101 Tampere, FINLAND, **BUTTERFLY UNIT SUPPORTING***

RADIX-4 AND RADIX-2 *FFT (2003).*

*[17] Yihu Xu; Chung-Hoon Lee; Myong-Seob Lim; , "**Design of split-radix FFT pruning for OFDM based cognitive radio system,**" Circuits and Systems (APCCAS), 2010 IEEE Asia Pacific Conference on , vol., no., pp.524-527, 6-9 Dec. 2010*

*[18] C. Vennila; C. T. Kumar Palaniappan; K. V. Krishna; G. Lakshminarayanan; Seok-Bum Ko "**Dynamic partial reconfigurable FFT/IFFT pruning for OFDM based Cognitive radio**"- 2012 IEEE International Symposium on Circuits and Systems (May 2012), pg. 33-36*

*[19] John D Markel, "**FFT Pruning**" IEEE Transactions on Audio and Electro acoustics, Vol -19 No.4, pp.305 – 311, December1971*

*[20] Yihu Xu; Daehwan Kim; SangHak Lee; Myong-Seob Lim; , "**Split-radix FFT pruning for OFDM based Cognitive Radio system**" Communications and Information Technology, 2009. ISCIT 2009. 9th International Symposium on , vol., no., pp.421-424, 28-30 Sept. 2009*

[21] Scaling Accumulator Multipliers http://www.fpga-guru.com/multipli.htm

*[22] Rakesh Rajbanshi, Alexander M. Wyglinski, and Gary J. Minden, "**An efficient implementation of nc-ofdm transceivers for cognitive radios**," in Cognitive Radio Oriented Wireless Networks and Communications, 2006. 1st International Conference on, june 2006, pp. 1 –5.*

*[23] In-Gul Jang; Zhe-Yan Piao; Ze-Hua Dong; Jin-Gyun Chung; Kang-Yoon Lee; , "**Low-power FFT design for NC-OFDM in cognitive radio systems,**" Circuits and Systems (ISCAS), 2011 IEEE International Symposium on , vol., no., pp.2449-2452, 15-18 May 2011*

[24] *http://en.wikipedia.org/wiki/Field-programmable_gate_array*

[25] *http://home.mit.bme.hu/~szedo/FPGA/fpgahw.htm*

[26] *Datasheet ALTERA FPGA STRATIX V*

[27] *Cordeiro, Carlos; Challapali, K.; Birru, D.; Sai Shankar, N., "**IEEE 802.22: the first worldwide wireless standard based on cognitive radios,**" New Frontiers in Dynamic Spectrum Access Networks, 2005. DySPAN 2005. 2005 First IEEE International Symposium on , vol., no., pp.328,337, 8-11 Nov. 2005*